Improve Agents without Retraining: Parallel Tree Search with Off-Policy Correction

Anonymous Author(s)
Affiliation
Address
email

Abstract

Tree Search (TS) is crucial to some of the most influential successes in reinforcement learning. Here, we tackle two major challenges with TS that limit its usability: *distribution shift* and *scalability*. We first discover and analyze a counter-intuitive phenomenon: action selection through TS and a pre-trained value function often leads to lower performance compared to the original pre-trained agent, even when having access to the exact state and reward in future steps. We show this is due to a distribution shift to areas where value estimates are highly inaccurate and analyze this effect using Extreme Value theory. To overcome this problem, we introduce a novel off-policy correction term that accounts for the mismatch between the pre-trained value and its corresponding TS policy by penalizing under-sampled trajectories. We prove that our correction eliminates the above mismatch and bound the probability of sub-optimal action selection. Our correction significantly improves pre-trained Rainbow agents without any further training, often more than doubling their scores on Atari games. Next, we address the scalability issue given by the computational complexity of exhaustive TS that scales exponentially with the tree depth. We introduce Batch-BFS: a GPU breadth-first search that advances all nodes in each depth of the tree simultaneously. Batch-BFS reduces runtime by two orders of magnitude and, beyond inference, enables also training with TS of depths that were not feasible before. We train DQN agents from scratch using TS and show improvement in several Atari games compared to both the original DQN and the more advanced Rainbow. We will share the code upon publication.

1 Introduction

Tree search (TS) is a fundamental component of Reinforcement Learning (RL) [41] used in some of the most successful RL systems [37, 39]. One popular TS method, Monte-Carlo Tree Search (MCTS) [10], achieved superhuman performance in board games like Go [39], Chess [40], and Bridge [5]. MCTS gradually unfolds the tree by adding nodes and visitation counts online and storing them in memory for future traversals. This paradigm is suitable for discrete state-spaces where counts are aggregated across multiple iterations as the tree is built node-by-node, but less suitable for continuous state-spaces or image-based domains like robotics and autonomous driving. For the same reason, MCTS cannot be applied for improving pre-trained agents without collecting their visitation statistics in training iterations.

Instead, we explore the possibility of conducting the TS "on-demand" by expanding the tree up to a given depth at each state. Our approach handles continuous and large state-spaces like images without requiring any memorization. This on-demand TS can be performed both at training or inference time. Here, we focus our attention on the second case, which leads to score improvement without any re-training. This allows one to better utilize existing pre-trained agents even without having the

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

ability or resources to train them. For example, a single AlphaGo training run is estimated to cost 35 million USD [1]. In other cases, even when compute budget is not a limitation, the setup itself makes training inaccessible. For example, when models are distributed to end-clients with too few computational resources to train agents in their local custom environments.

We run TS for inference as follows. For action selection, we feed the states at the leaves of the spanned tree to the pre-trained value function. We then choose the action at the root according to the branch with the highest discounted sum of rewards and value at the leaves. Our approach instantly improves the scores of agents that were already trained for long periods (see Sec. 5.1). Often, such improvement is possible because the value function is not fully realizable with a function approximator, e.g., a deep neural network; TS can then overcome the limitation of the model. In practice, TS requires access to a forward model that is fed with actions to advance states and produce rewards. Here, we build on the recently published CuLE [11] – an Atari emulator that runs on GPU. This allows us to isolate the fundamental properties of TS without the added noise of learned models such as those described in Sec.6.

Performing TS on-demand has many benefits, but it also faces limitations. We identify and analyze two major obstacles: distribution shift and scalability.

First, we report a counter-intuitive phenomenon when applying TS to pre-trained agents. As TS looks into the future, thus utilizing more information from the environment, one might expect that searching deeper should yield better scores. Surprisingly, we find that in many cases, the opposite happens: action selection based on vanilla TS can drastically impair performance. We show that performance deteriorates due to a distribution shift from the original pre-trained policy to its corresponding tree-based policy. We analyze this phenomenon by quantifying the probability of choosing a sub-optimal action when the value function error is high. This occurs because for values of out-of-distribution states, larger variance translates to a larger bias of the maximum. Our analysis leads to a simple, computationally effective off-policy correction term based on the Bellman error. We refer to the resulting TS as the Bellman-Corrected Tree-Search (BCTS) algorithm. BCTS yields monotonically improving scores as the tree depth increases. In several Atari games, BCTS even more than doubles the scores of pre-trained Rainbow agents [20].

The second limitation is scalability: the tree grows exponentially with its depth, making the search process computationally intensive and limiting the horizon of forward-simulation steps. To overcome this limitation, we propose Batch-BFS: a parallel GPU adaptation of Breadth-First Search (BFS), which brings the runtime down to a practical regime. We measured orders-of-magnitude speed-up compared to alternative approaches. Thus, in addition to improving inference, it also enables training tree-based agents in the same order of training time without a tree. By combining Batch-BFS with DQN [28] and training it with multiple depths, we achieve performance comparable or superior to Rainbow – one of the highest-scoring variants of the DQN-based algorithm family.

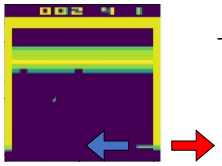
Our Contributions. (1) We identify and analyze a distribution-shift that impairs post-training TS. (2) We introduce a correction mechanism and use it to devise BCTS: an efficient algorithm that improves pre-trained agents, often doubling their scores or more. (3) We create Batch-BFS, an efficient TS on GPU. (4) We use Batch-BFS to train tree-based DQN agents and obtain higher scores than DQN and Rainbow.

2 Preliminaries

Our framework is an infinite-horizon discounted Markov Decision Process (MDP) [34]. An MDP is defined as the 5-tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is a state space, $\mathcal{A}$ is a finite action space, $P(s'|s,a)$ is a transition kernel, $r(s,a)$ is a reward function, $\gamma \in (0,1)$ is a discount factor. At each step $t = 0, 1, \ldots$, the agent observes the last state s_t, performs an action a_t and receives a reward r_t. The next state is then sampled by $s_{t+1} \sim P(\cdot|s_t, a_t)$. For brevity, we denote $A := |\mathcal{A}|$.

Let $\pi : \mathcal{S} \to \mathcal{A}$ be a stationary policy. Let $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be the state-action value of a policy π, defined in state s as $Q^\pi(s,a) \equiv \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) \middle| s_0 = s, a_0 = a\right]$, where $\mathbb{E}^\pi$ denotes expectation w.r.t. the distribution induced by π. Our goal is to find a policy π^* yielding the optimal value Q^* such that $Q^*(s,a) = \max_\pi r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \max_{a'} Q^\pi(s',a')$. It is well known that

$$Q^*(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \max_{a'} Q^*(s',a'), \qquad \pi^*(s) = \arg\max_a Q^*(s,a).$$



a	$Q(s_0, a)$	$Q(s_1, a)$	$Q(s_1, a)$	$Q(s_1, a)$	$Q(s_1, a)$
Left	2.6103	2.7906	2.8114	2.8114	2.8326
No-op	2.5712	2.7874	2.8082	2.8082	2.8185
Fire	2.5801	2.7837	2.8041	2.8041	2.8192
Right	2.3896	2.7748	2.7914	2.7914	2.8022

Figure 1: **A failure of vanilla tree search.** *Left*: An Atari Breakout frame. *Right*: Q-values of TS for the frame on the left. Rows correspond to the action taken at the considered depth, which is $d = 0$ for the first column and $d = 1$ for the four others. The action at the root of the tree is color-coded: Red for 'Right', and blue for 'Left'.

Vanilla tree search. To ease notations and make the results concise, we limit the analysis to deterministic transitions[1], i.e., an action sequence $(a_0, \dots, a_{d-1})$, starting at s_0 leads to a corresponding trajectory $(s_0, \dots, s_d)$. Nonetheless, the results can be extended to a stochastic setup by working with the marginal probability over the trajectory. Then, for a policy π_o, let the d-step Q-function

$$Q_d^{\pi_o}(s,a) = \left[\max_{(a_k)_{k=1}^d \in \mathcal{A}} \left[\sum_{t=0}^{d-1} \gamma^t r(s_t, a_t) \right] + \gamma^d Q^{\pi_o}(s_d, a_d) \right]_{s_0=s, a_0=a}, \tag{1}$$

and similarly let $\hat{Q}_d^{\pi_o}(s,a)$ be the d-step Q-function estimator that uses an estimated Q-function $\hat{Q}^{\pi_o}$ instead of Q^{π_o}. Finally, denote by π_d the d-step greedy policy

$$\pi_d(s) := \arg\max_{a \in \mathcal{A}} \hat{Q}_d^{\pi_o}(s,a). \tag{2}$$

3 Solving the Tree Search Distribution Shift

In this section, we show how to leverage TS to improve a pre-trained policy. We start by demonstrating an intriguing phenomenon: The quality of agents degrades when using vanilla TS. We then analyze the problem and devise a solution.

3.1 Performance degradation with vanilla tree search

We focus on applying TS at inference time, without learning. We begin with a simple experiment that quantifies the benefit of using TS given a pre-trained policy.

A TS policy has access to future states and rewards and, by definition, is optimal when the tree depth goes to infinity. Hence, intuitively, one may expect a TS policy to improve upon π_o for finite depths as well. To test this, we load Rainbow agents $\hat{Q}^{\pi_o}$, pre-trained on 50M frames; they are publicly available in [22] and achieve superhuman scores as in [20]. We use them to test TS policies π_d with multiple depths d on several Atari benchmarks. Surprisingly, the results (red curves in Fig. 4, Sec. 5.1) show that TS reduces the total reward, sometimes to scores of random policies, in various games and depths. The drop is particularly severe for TS policies with $d = 1$ — a fact later explained by our analysis in Thm. 3.5.

We find the reason for this performance drop to be the poor generalization of the value function to states outside the stationary π_o's distribution. Fig. 1 shows a typical, bad action selection in Atari Breakout by a depth-1 TS. The table on the right reports the estimated Q-values of the root state (first column) and of every state at depth 1 (last four columns). Since the ball is dropping, 'Left' is the optimal action. Indeed, this corresponds with the Q-values at the root. However, at depth 1, the Q-values of the future state that corresponds to choosing 'Left' (second column) are the lowest among all depth-1 future states. Subsequently, the depth-1 TS policy selects 'Right'.

During training, towards convergence, the trained policy mostly selects 'Left' while other actions are rarely sampled. Therefore, expanding the tree at inference time generates states that have been hardly

[1] The Atari environments we experiment on here are indeed close to deterministic.

observed during training and are consequently characterized by inaccurate Q-value estimates. In the case of Fig. 1, the Q-value for 'Left' should indeed be low because the agent is about to lose the game. As for the other, poorly sampled states, regression towards a higher mean leads to over-estimated Q-values. Similar poor generalization has been observed in previous studies [16] and is interpreted as an off-policy distribution shift [30].

Beyond the anecdotal example in Fig. 1, additional evidence support our interpretation regarding the distribution shift. We first consider the error in the value estimate captured by the Bellman error minimized in training. We compare the average Bellman error of the action chosen by π_o to all other actions at the tree root. When averaging over 200 episodes, we find that the error for actions chosen by π_o is consistently lower than for other actions: $\times 1.5$ lower for Breakout, and $\times 2$ for Frostbite. We also measure the off-policy distribution shift between π_o and the TS policy by counting disagreements on actions between the two policies. In Breakout, π_o and π_1 agreed only in 18% of the states; in Frostbite, the agreement is only 1.96%. Such a level of disagreement between a well-trained agent and its one-step look-ahead extension is surprising. On the other hand, it accounts for the drastic drop in performance when applying TS, especially in Frostbite (Fig. 4).

3.2 Analysis of the degradation

We analyze the decision process of a policy π_d, given a pre-trained value function estimator, in a probabilistic setting. Our analysis leads to a simple correction term given at the end of the section. We also show how to compute this correction from the TS. Formally, we are given a policy represented as a value function $\hat{Q}^{\pi_o}$, which we feed to the d-step greedy policy (2) for each action selection. Policy training is a stochastic process due to random start states, exploration, replay buffer sampling, etc. The value estimator $\hat{Q}^{\pi_o}$ can thus be regarded as a random variable, with an expectation that is a deterministic function Q^{π_o} with a corresponding policy π_o, where 'o' stands for 'original'.

In short, we bound the probability that a sub-optimal action falsely appears more attractive than the optimal one. We thus wish to conclude with high probability whether $a_0 = \arg\max_a \hat{Q}_d^{\pi_o}(s,a)$ is indeed optimal, i.e., $Q_d^{\pi_o}(s,a_0) \geq Q_d^{\pi_o}(s,a)\ \forall a \in \mathcal{A}$.

As we have shown in Sec. 3.1 that states belonging to trajectories that follow π_o have lower value estimation noise, we model this effect via the following two assumptions. Here, we denote by $t=0$ the time when an agent acts and not as the time step of the episode.

Assumption 1. *Let $\sigma_o, \sigma_e \in \mathbb{R}^+$ s.t. $0 < \sigma_o < \sigma_e$. For action sequence $(a_0, \dots, a_{d-1})$ and corresponding state trajectory $(s_0, \dots, s_d)$,*

$$\hat{Q}^{\pi_o}(s_d, a_d) \sim \begin{cases} \mathcal{N}(Q^{\pi_o}(s_d,a_d), \sigma_o^2) & \textit{if } a_0 = \pi_o(s_0) \\ \mathcal{N}(Q^{\pi_o}(s_d,a_d), \sigma_e^2) & \textit{otherwise}. \end{cases}$$

Assumption 1 presumes that a different choice of the first action a_0 yields a different last state s_d. This is commonly the case for environments with large state spaces, especially when stacking observations as done in Atari. While assuming a normal distribution is simplistic, it still captures the essence of the search process. Regarding the expectation, recall that π_o is originally obtained from the previous training stage via gradient descent with a symmetric loss function $\mathcal{L}$. Due to the symmetric loss, the estimate $\hat{Q}_\theta^{\pi_o}$ is unbiased, i.e., $\mathbb{E}[\hat{Q}_\theta^{\pi_o}] = Q^{\pi_o}$. Regarding the variance, towards convergence, the loss is computed on replay buffer samples generated according to the stationary distribution of π_o. The estimate of the value function for states outside the stationary distribution is consequently characterized by a higher variance, i.e. $\sigma_o < \sigma_e$. We also show that this separation of variance occurs in the data, as detailed in the last paragraph of Sec. 3.1.

After conditioning the variance on whether π_o was followed, we similarly split the respective sub-trees. To split, we assume the cumulative reward along the tree and values at the leaves depend only on (i) the root state and (ii) whether π_o was selected at that state.

Assumption 2. *For action sequence $(a_0, \dots a_{d-1})$ and corresponding trajectory $(s_0, \dots, s_d)$,*

$$\sum_{t=0}^{d-1} \gamma^t r(s_t, a_t) = \begin{cases} R_o(s_0) \textit{ if } a_0 = \pi_o(s_0) \\ R_e(s_0) \textit{ otherwise}, \end{cases} \qquad Q^{\pi_o}(s_d, a_d) = \begin{cases} \mu_o(s_0) \textit{ if } a_0 = \pi_o(s_0) \\ \mu_e(s_0) \textit{ otherwise}, \end{cases}$$

with R_o, R_e, μ_o, μ_e being functions of s.

Assumption 2 could be replaced with a more detailed consideration of each trajectory, but we make it for simplicity. This assumption considers the worst-case scenario: the rewards are unhelpful in determining the optimal policy, and all leaves are equally likely to mislead the d-step greedy policy.

Assuming from now on that Assumptions 1 and 2 hold, we can now explicitly express the distribution of the maximal value among the leaves using Generalized Extreme Value (GEV) theory [9].

Lemma 3.1. *It holds that*

$$\hat{Q}_d^{\pi_o}(s,\pi_o(s)) = R_o(s) + \gamma^d G_o(s), \qquad \max_{a\neq\pi_o(s)} \hat{Q}_d^{\pi_o}(s,a) = R_e(s) + \gamma^d G_e(s),$$

with

$$G_o(s) \sim GEV(\mu_o^{GEV}(s), \sigma_o^{GEV}, 0), \quad G_e(s) \sim GEV(\mu_e^{GEV}(s), \sigma_e^{GEV}, 0),$$

where GEV is the Generalized Extreme Value distribution and $\mu_o^{GEV}(s), \sigma_o^{GEV}, \mu_e^{GEV}(s), \sigma_e^{GEV}$ *are given in Appendix A.2.*

All proofs are deferred to Appendix A. Using the GEV distribution, we can now quantify the bias stemming from the maximization in each of two sub-trees corresponding π_o vs. all other actions.

Lemma 3.2. *It holds that*

$$\mathbb{E}\left[\hat{Q}_d^{\pi_o}(s,\pi_o(s))\right] = Q_d^{\pi_o}(s,\pi_o(s)) + \gamma^d B_o(\sigma_o, A, d)$$

$$\mathbb{E}\left[\max_{a\neq\pi_o(s)} \hat{Q}_d^{\pi_o}(s,a)\right] = Q_d^{\pi_o}(s,a\neq\pi_o(s)) + \gamma^d B_e(\sigma_e, A, d),$$

where the biases $B_o,\ B_e$ *are given in Appendix A.2, and satisfy* $0 < B_o(\sigma_o, A, d) < B_e(\sigma_e, A, d)$.

Lemma 3.2 conveys the main message of our analysis: the variance of terms being maximized translates to a positive shift in the expectation of the maximum. Hence, even if $\mu_o(s) > \mu_e(s)$ for a certain s, a different action than $\pi_o(s)$ can be chosen with non-negligible probability as the bias in $\mu_e(s)$ is greater than the one in $\mu_o(s)$. To compensate for this bias that gives an unfair advantage to the noisier nodes of the tree, we introduce a penalty term that precisely cancels it.

3.3 BCTS: Bellman Corrected Tree Search

Instead of selecting actions via (2), in BCTS we replace $\hat{Q}_d^{\pi_o}$ with the corrected Q_d^{BCTS} defined by

$$\hat{Q}_d^{\text{BCTS},\pi_o}(s,a) := \begin{cases} \hat{Q}_d^{\pi_o}(s,a) & \text{if } a_0 = \pi_o(s_0), \\ \hat{Q}_d^{\pi_o}(s,a) - \gamma^d \left(B_e(\sigma_o, A, d) - B_o(\sigma_e, A, d)\right) & \text{otherwise,} \end{cases} \tag{3}$$

and we denote the BCTS policy by

$$\pi_d^{\text{BCTS}} := \arg\max_a \hat{Q}_d^{\text{BCTS},\pi_o}(s,a). \tag{4}$$

In the following result, we prove that BCTS indeed eliminates undesirable bias.

Theorem 3.3. *The relation* $\mathbb{E}\left[\hat{Q}_d^{BCTS,\pi_o}(s,\pi_o(s))\right] > \mathbb{E}\left[\hat{Q}_d^{BCTS,\pi_o}(s,a)\right],\ a\neq\pi_o(s)$, *holds if and only if* $Q_d^{\pi_o}(s,\pi_o(s)) > Q_d^{\pi_o}(s,a\neq\pi_o(s))$.

The biases $B_o,\ B_e$ include the inverse of the cumulative standard normal distribution Φ^{-1} (Appendix A.2). We now approximate them with simple closed-form expressions that are highly accurate for $d \geq 2$ (Appendix A.4). These approximations help revealing how the problem parameters dictate prominent quantities such as the correction term in (3) and the probability in Thm. 3.5 below.

Lemma 3.4. *When* $A^{d-1} \gg 1$, *the correction term in* (3) *can be approximated with*

$$B_e(\sigma_o, A, d) - B_o(\sigma_e, A, d) \approx \sqrt{2\log A}\left(\sigma_e\sqrt{d} - \sigma_o\sqrt{d-1}\right). \tag{5}$$

The bias gap in (5) depends on the ratio between σ_e and σ_o; this suggests that TS in different environments will be affected differently. As $\sigma_e > \sigma_o$, the bias gap is positive. This is indeed

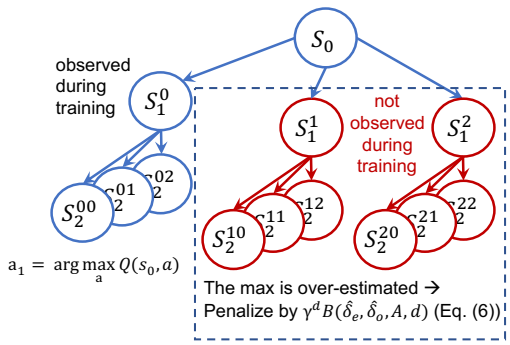


Figure 2: **BCTS Algorithm.** Exploring the full tree reveals out-of-distribution states (red). These were not visited during training and tend to have highly variable scores, leading to high overestimation error. The penalty term $B(\hat{\delta}_e, \hat{\delta}_o, A, d)$ (see (6)) cancels the excess bias. The Bellman errors $\hat{\delta}_e, \hat{\delta}_o$ are extracted from the tree at depth 1.

expected, since the maximum over the sub-trees of $a_0 \neq \pi_o(s)$ includes noisier elements than those in the sub-tree of $a_0 = \pi_o(s)$. Also, in (5), $\sigma_e\sqrt{d}$ dominates $\sigma_o\sqrt{d-1}$, making the bias gap grow asymptotically with $\sqrt{d \log A}$. This rate reflects how the number of elements being maximized over affects the bias of their maximum.

Next, we bound the probability of choosing a sub-optimal action when using BCTS. In Appendix A, Thm. A.1, we give an exact bound to that probability without assuming $A^{d-1} \gg 1$. Here, we apply Lemma. 3.4 to give the result in terms of $\sigma_o,\ \sigma_e$.

Theorem 3.5. *When $A^{d-1} \gg 1$, the policy $\pi_d^{BCTS}(s)$ (see (4)) chooses a sub-optimal action with probability bounded by:*

$$\Pr\left(\pi_d^{BCTS}(s) \notin \arg\max_a Q_d^{\pi_o}(s,a)\right) \leq \left(1 + \frac{6d\log A\left(Q_d^{\pi_o}(s,\pi_o(s)) - Q_d^{\pi_o}(s, a \neq \pi_o(s))\right)^2}{\gamma^{2d}\pi^2\left(\sigma_o^2 + \sigma_e^2\right)}\right)^{-1}.$$

The fraction in the above bound is a signal-to-noise ratio: The expected value difference in the numerator represents the signal, while the variances in the denominator represent the noise. In addition, the signal is "amplified" by $d \log A$ because, after applying the correction from (3), a larger number of tree nodes amount to a more accurate maximum estimator. A similar amplification also occurs due to nodes being deeper and is captured by γ^d in the denominator.

The factor γ^d also appears in the correction term from (3). This correction is the bias gap from (5), scaled by γ^d. Hence, asymptotically, while the bias gap grows with $\sqrt{d}$, the exponential term is more dominant; so, overall, this product decays with d. This decay reduces the difference between the vanilla and BCTS policies for deeper trees by lowering the portion of the estimated value compared to the exact reward. As we show later, this phenomenon is consistent with our experimental observations.

Computing the correction term requires estimates of σ_o and σ_e. As a surrogate for the error, we use the Bellman error. To justify it, let us treat the values of $\hat{Q}_d^{\pi_o}$ at different depths as samples of $\hat{Q}^{\pi_o}$. Then, the following result holds for its variance estimator. Note that we do not need Assumptions 1 and 2 to prove the following result.

Proposition 3.6. *Let $\hat{var}_n[X]$ be the variance estimator based on n samples of X. Then,*

$$\hat{var}_{n=2}[\hat{Q}^{\pi_o}(s,a)] = \left(\hat{Q}_1^{\pi_o}(s,a) - \hat{Q}_0^{\pi_o}(s,a)\right)^2/2 = \delta^2(s,a)/2,$$

where $\delta(s,a)$ is the Bellman error.

Note that during a TS, at depth 1 we have access to $\delta(s_0, a)$ of all $a \in \mathcal{A}$ without additional computation. For depths 2 and above, the Bellman error is defined only for actions chosen by π_o, corresponding to a single trajectory down the tree. For these reasons, we base the above result on samples from depths 0 and 1.

Thanks to Prop. 3.6, we can estimate the bias correction term in Lemma 3.4 directly from the TS operation. The resulting correction term is

$$B(\hat{\delta}_e, \hat{\delta}_o, A, d) = \sqrt{\log A}\left(\hat{\delta}_e\sqrt{d} - \hat{\delta}_o\sqrt{d-1}\right), \tag{6}$$

where $\hat{\delta}_o$ is the Bellman error corresponding to $a = \pi_o(s)$ at the root, and $\hat{\delta}_e$ and is the average Bellman error of all other actions. A visualization of the resulting BCTS algorithm is in Fig. 2.

Algorithm 1 Batch-BFS

Input: GPU environment $\mathcal{G}$, value network Q_θ, depth d
Init tensors: state $\bar{S} = [s_0]$, action $\bar{A} = [0, 1, 2, .., A-1]$, reward $\bar{R} = [0]$
for $i_d = 0$ **to** $d-1$ **do**
  $\bar{S} \leftarrow \bar{S} \times A, \;\; \bar{R} \leftarrow \bar{R} \times A$ // Replicate state and reward tensors A times
  $\bar{r}, \bar{S}' = \mathcal{G}([\bar{S}, \bar{A}])$ // Feed $[\bar{S}, \bar{A}]$ to simulator and advance
  $\bar{R} \leftarrow \bar{R} + \gamma^{i_d}\bar{r}, \;\; \bar{S} \leftarrow \bar{S}'$ // Accumulate discounted reward
  $\bar{A} \leftarrow \bar{A} \times A$ // Replicate action tensor A times
end for
$\bar{R} \leftarrow \bar{R} + \gamma^d \max_a Q_\theta(\bar{S}, a)$ // Accumulate discounted value of states at depth d
Return $\lfloor (\arg\max \bar{R})/A^{d-1} \rfloor$ // Return optimal action at the root

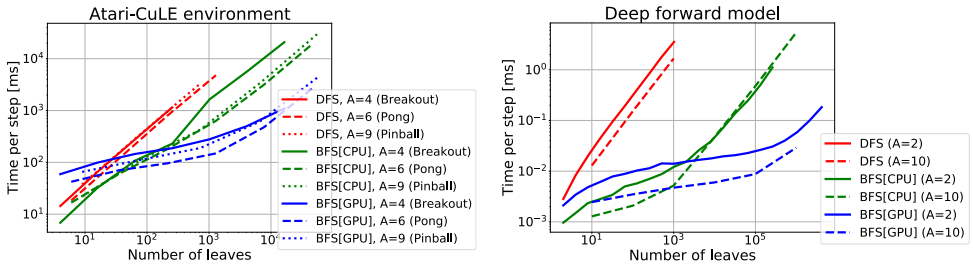


Figure 3: **Average tree-search time per action selection**. *Left:* Atari-CuLE Breakout, Pong, and VideoPinball. *Right:* A randomly generated neural network to mimic a learned forward-model with $A \in \{2, 10\}$. Note that x and y axes are in log-scale.

4 Solving Scalability via Batch-BFS

The second drawback of TS is scalability: exhaustive TS is impractical for non-trivial tree depths because of the exponential growth of the tree dimension. As TS requires generating $|A|^d$ leaves at depth d, it has been rarely considered as a viable solution. To solve this issue, we propose Batch-BFS, an efficient, parallel, TS scheme based on BFS, built upon the ability to advance multiple environments simultaneously; see Alg. 1. It achieves a significant runtime speed-up when a forward model is implemented on a GPU. Such GPU-based environments are becoming common nowadays because of their advantages (including parallelization and higher throughput) over their CPU counterparts. E.g., Isaac-Gym [32] provides a GPU implementation robotic manipulation tasks [41], whereas Atari-CuLE [11] is a CUDA-based version of the AtariGym benchmarks [4]. Batch-BFS is not limited to exact simulators and can be applied to learned deep forward models, like those in [31, 21]. Since Batch-BFS simultaneously advances the entire tree, it enables exhaustive tree expansion to previously infeasible depths. It also allows access to the cumulative reward and estimated value over all the tree nodes. Such access to all future values paves a path to new types of algorithms for, e.g., risk reduction and early tree-pruning. We leave such directions to future work. In addition to Alg. 1, we also provide a visualization of Batch-BFS in Appendix B, Fig. 5.

4.1 Runtime experiments

To showcase the efficacy of Batch-BFS, we compare it to a CPU-based BFS and to non-parallel TS, i.e., Depth-First-Search (DFS). We measure the duration of an average single TS operation on two environments: Atari-CuLE [11] and a Deep NN (DNN) mimicking a learned forward model. The DNN is implemented in pytorch and uses cudnn [8]. It consists of three randomly-initialized hidden layers of width 100 with input size 100 for the state and 2 or 10 for the actions. The results are given in Fig. 3. We run our experiments on a 8 core Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz equipped with one NVIDIA Tesla V100 16GB. Although we used a single GPU for our experiments, we expect a larger parallelization (and thus computational efficiency) to be potentially achieved in the future through a multi-GPU implementation. As expected, DFS scales exponentially in depth and is slower than BFS. When comparing BFS on CPU vs. GPU, we see that CPU is more efficient in low depths. This is indeed expected, as performance of GPUs without parallelization is inferior to that of

CPUs. This issue is often addressed by distributing the simulators across a massive number of CPU cores [15]. We leverage this phenomenon in Batch-BFS by finding the optimal "cross-over depth" per game and swap the compute device in the middle of the search from CPU to GPU.

5 Experiments

In this section, we report our results on two sets of experiments: the first deals solely with TS for inference, without learning, whereas the second includes the case of TS used in training.

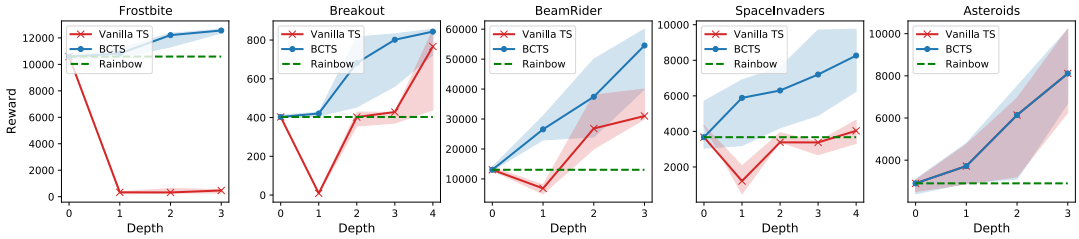


Figure 4: **Inference only: Vanilla TS vs. BCTS.** Median scores with lower (0.25) and upper (0.75) quantiles over 200 episodes, as a function of the tree depth. Surprisingly, a deeper tree often degrades the performance of vanilla TS. BCTS (blue) improves upon vanilla TS (red) for all depths, except for Asteroids. The improvement grows monotonically with the tree depth.

5.1 Inference with tree search

Using the pre-trained Rainbow agents in [22] (see Sec. 3), we test vanilla TS and show (Fig. 4, red plots) that it leads to a lower score than the baseline π_o. The largest drop is for $d = 1$ as supported by our analysis. As for BCTS, we found that multiplying its correction term (see (6)) by a constant that we sweep over can improve performance; we applied this method for the experiments here. Recall that BCTS is a TS applied on the pre-trained Rainbow baseline; i.e., the case of $d = 0$ is Rainbow itself. The results in Fig. 4 show that BCTS significantly improves the scores, monotonically in depth, in all games. It improves the Rainbow baseline already for $d = 1$, while for $d = 4$, the score more than doubles. For BeamRider, BCTS with $d = 4$ achieves roughly $\times 5$ improvement. Notice that without the computationally efficient implementation of Batch-BFS, the results for deeper trees would not have been obtainable in a practical time. We provide timing measurements per game and depth in Appendix C.1. Finally, notice that the advantage provided by BCTS is game-specific. Different games benefit from it by a different amount. In one of the games tested, Asteroids, vanilla TS was as useful as BCTS. Our findings reported in the last paragraph of Sec. 3.1 hint why certain games benefit from BCTS more than others. However, a more thorough study on how the dynamics and state distribution in different games affect TS constitutes an interesting topic for future research.

5.2 Training with tree search

To further demonstrate the potential benefits of TS once a computationally efficient implementation (Section 4) is available, we show how it affects training agents from scratch on CuLE-Atari environments. We extend classic DQN [28] with TS using Batch-BFS for each action selection. Notice that training with TS does not suffer from the distribution shift studied in Sec. 3. Hence, the experiments below use vanilla TS and not BCTS.

Our experiment is even more significant considering that Efroni et al. [14] recently proved that the Bellman update should be replaced so that contraction is guaranteed for tree-based policies only when the value at the leaves is backed. However, this theory was not supported by empirical evidence beyond a toy maze. As far as we know, our work is the first to adopt this corrected update to obtain favorable results in state-of-the-art domains, thanks to the computationally efficient TS implementation achieved by Batch-BFS. We find this method to be beneficial in several of the games we tested. In the experiments below, we treat the correction from [14] as a hyper-parameter and include ablation studies of it in Appendix C.3.

Table 1: **Atari scores after two days of training.** We follow the evaluation method in [20]: average of 200 testing episodes, from the agent snapshot that obtained the highest score during training.

Game	DQN with TS, depth d				DQN [28]	Rainbow[20]
	$d = 1$	$d = 2$	$d = 3$	$d = 4$		
Asteroids	$2,093$	$2,613$	$4,794$	$\mathbf{17,929}$	$1,664$	$1,594$
Breakout	385	581	420	$\mathbf{620}$	377	327
MsPacman	$1,644$	$2,923$	$3,498$	$\mathbf{4,021}$	$2,398$	$3,600$
SpaceInvaders	675	$1,602$	$2,132$	$\mathbf{2,550}$	$1,132$	$2,162$
VideoPinball	$229,129$	$244,052$	$442,347$	$345,742$	$163,720$	$\mathbf{641,235}$

We show the training scores in Table 1 and convergence plots in Appendix C.2. For a fair comparison of different TS depths, we stop every run after two days on the same hardware (see Appendix E), not considering the total iteration count. To compare our results against classic DQN and Rainbow, we measure the number of iterations completed in two days by DQN with TS, $d = 0$. In Table 1 we report the corresponding intermediate results for DQN and Rainbow reported by the original authors in [35]. In most games, it amounts to roughly 30 million iterations. As already shown in the case of inference with no training, the achieved score increases monotonically with the tree depth. In four of the five games, DQN with TS even surpasses the more advanced Rainbow. Since all results were obtained for identical runtime, improvement per unit of time is higher for higher depths. This essentially translates to better efficiency of compute resources. Convergence plots as a function of wall-clock time are shown in Appendix D. We also tested TS on two additional games not included in Table 1: Boxing and Pong. Interestingly, TS with $d = 4$ immediately obtained the highest possible scores in both these games already in the first training iteration.

6 Related work

The idea of searching forward in time has been employed extensively in control theory via methods such as A* [19], RRT [26], and MPC [2]. The latter is quite popular in the context of efficient planning in RL [31, 42, 43]. MPC-based controllers rely on recourse planning by solving an optimization program given the continuous structure of the dynamics. In our setup, the controls are discrete and the forward model is a black-box that cannot be directly used in an optimization scheme.

We leverage an efficient GPU simulator to conduct the look-ahead search. When such a simulator is out of reach, it can be learned from data. There is vast literature regarding learned deep forward models without optimizing a policy [24, 33, 25]. Other works [23, 21, 18, 36] used the learned model for planning but not with a TS. Few considered roll-outs of learned dynamics [6, 17] but only for evaluation purposes. Additional relevant works are MuZero [37] and "MuZero Unplugged" [38] which utilized a learned forward-model for prediction in an MCTS-based policy. In [29], the trade-off between learning and planning using a TS was empirically tested. Finally, look-ahead policies for RL were also studied theoretically; bounds on the suboptimality of the learned policy were given in [12, 13, 14]. There, the focus was on the effect of planning on the learning process.

7 Discussion

Our study of the degradation with vanilla TS implies that the learned value function is not representative of the actual states that can be visited. This conclusion can be helpful in debugging RL systems. It can also be used to better tune the approximation architecture, or guide exploration.

Our solution to the above performance degradation is an off-policy correction that penalizes high-error trajectories. It can be further improved with different notions of uncertainty for the value function, e.g., Bayesian or bootstrapped models. Also, while some simulators are available on GPU, such as Atari [11] and robotic manipulation [27], in other cases, a learned model can be used. In these cases, one could include the simulator quality in the off-policy penalty term. Finally, a limitation of our work is that we focus on problems with a discrete action space. Handling problems with continuous action tasks is a challenging direction for future work.

Broader Impact. The paper proposes a method to improve existing RL algorithms. As such, its main impact is to make RL more easily and widely deployable. Since our method can be applied to policies trained with any algorithm, it can be viewed as a generic "policy booster", and find applications with access to the environment model or its approximation.

References

[1] How much did alphago zero cost? https://www.yuzeh.com/data/agz-cost.html. Accessed: 2021-05-20.

[2] Paul Serban Agachi, Mircea Vasile Cristea, Alexandra Ana Csavdari, and Botond Szilagyi. *2. Model predictive control*. De Gruyter, 2016.

[3] JM Blair, CA Edwards, and JH Johnson. Rational chebyshev approximations for the inverse of the error function. *Mathematics of Computation*, 30(136):827–830, 1976.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[5] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[6] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *arXiv preprint arXiv:1807.01675*, 2018.

[7] Francesco Paolo Cantelli. Sui confini della probabilita. In *Atti del Congresso Internazionale dei Matematici: Bologna del 3 al 10 de settembre di 1928*, pages 47–60, 1929.

[8] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning, 2014. cite arxiv:1410.0759.

[9] Stuart Coles, Joanna Bawa, Lesley Trenner, and Pat Dorazio. *An introduction to statistical modeling of extreme values*, volume 208. Springer, 2001.

[10] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[11] Steven Dalton, Iuri Frosio, and Michael Garland. Accelerating reinforcement learning through gpu atari emulation. *arXiv preprint arXiv:1907.08467*, 2019, BSD 3-Clause "New" or "Revised" License.

[12] Yonathan Efroni, Gal Dalal, Bruno Scherrer, and Shie Mannor. Beyond the one-step greedy approach in reinforcement learning. In *International Conference on Machine Learning*, pages 1387–1396. PMLR, 2018.

[13] Yonathan Efroni, Gal Dalal, Bruno Scherrer, and Shie Mannor. Multiple-step greedy policies in approximate and online reinforcement learning. In *NeurIPS*, 2018.

[14] Yonathan Efroni, Gal Dalal, Bruno Scherrer, and Shie Mannor. How to combine tree-search methods in reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI 2019)*, 2019.

[15] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

[16] Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

[17] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

[18] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[19] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[20] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

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

[22] Kaixhin. Rainbow Pre-trained Agents v1.3, 2019, MIT License.

[23] Gabriel Kalweit and Joschka Boedecker. Uncertainty-driven imagination for continuous deep reinforcement learning. In *Conference on Robot Learning*, pages 195–206. PMLR, 2017.

[24] Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. Learning to simulate dynamic environments with gamegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1231–1240, 2020.

[25] Rahul G Krishnan, Uri Shalit, and David Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

[26] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. *Technical Report. Computer Science Department, Iowa State University*, 1998.

[27] Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. In *Conference on Robot Learning*, pages 270–282. PMLR, 2018.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[29] Thomas M Moerland, Anna Deichler, Simone Baldi, Joost Broekens, and Catholijn M Jonker. Think too fast nor too slow: The computational trade-off between planning and reinforcement learning. *arXiv preprint arXiv:2005.07404*, 2020.

[30] Remi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[31] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[32] NVIDIA. Isaac Gym preview release, 2021.

[33] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. *arXiv preprint arXiv:1507.08750*, 2015.

[34] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[35] John Quan and Georg Ostrovski. DQN Zoo: Reference implementations of DQN-based agents, 2020, Apache License 2.0.

[36] Sébastien Racanière, Théophane Weber, David P Reichert, Lars Buesing, Arthur Guez, Danilo Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5694–5705, 2017.

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

[38] Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. Online and offline reinforcement learning by planning with a learned model. *arXiv preprint arXiv:2104.06294*, 2021.

[39] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[40] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

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

[42] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721. IEEE, 2017.

[43] Mario Zanon and Sébastien Gros. Safe reinforcement learning using robust mpc. *IEEE Transactions on Automatic Control*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Assumptions 1, 2, and Sec. 7.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See the last paragraph in Section 7.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See the preliminaries in Section 2 and Assumptions 1, 2.
   (b) Did you include complete proofs of all theoretical results? [Yes] All proofs are given in Appendix A.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code-base relies on a public repository available online [22] with proper citation in Section 5. The algorithm and instructions for our experiments are given in Sec. 3.3, Section 4 and Section 5. Lastly, we started the legal process of publishing our adaptations to the public repository [22] and will share them publicly upon acceptance.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] All training details are given in Appendix C.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] Available in the corresponding references.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No] We started the legal process of releasing our extensions to the public repositories we rely on, and will publicly share them as new repositories upon acceptance.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]