

Waypoint-Based Reinforcement Learning for Robot Manipulation Tasks

Shaunak A. Mehta, Soheil Habibian, and Dylan P. Losey

Abstract—Robot arms should be able to learn new tasks. One framework here is reinforcement learning, where the robot is given a reward function that encodes the task, and the robot autonomously learns actions to maximize its reward. Existing approaches to reinforcement learning often frame this problem as a Markov decision process, and learn a policy (or a hierarchy of policies) to complete the task. These policies reason over hundreds of fine-grained actions that the robot arm needs to take: e.g., moving slightly to the right or rotating the end-effector a few degrees. But the manipulation tasks that we want robots to perform can often be broken down into a small number of high-level motions: e.g., reaching an object or turning a handle. In this paper we therefore propose a *waypoint-based* approach for model-free reinforcement learning. Instead of learning a low-level policy, the robot now learns a trajectory of waypoints, and then interpolates between those waypoints using existing controllers. Our key novelty is framing this waypoint-based setting as a sequence of multi-armed bandits: each bandit problem corresponds to one waypoint along the robot’s motion. We theoretically show that an ideal solution to this reformulation has lower regret bounds than standard frameworks. We also introduce an approximate posterior sampling solution that builds the robot’s motion one waypoint at a time. Results across benchmark simulations and two real-world experiments suggest that this proposed approach learns new tasks more quickly than state-of-the-art baselines. See our website here: <https://collab.me.vt.edu/r1-waypoints/>

I. INTRODUCTION

Robots often need to learn behaviors that optimize a reward function. For instance, in Figure 1 the robot’s objective is to open a drawer. To learn how to open this drawer the robot rolls-out behaviors in the environment and determines which actions increase its reward (i.e., which actions open the drawer). Existing approaches often solve this reinforcement learning problem by constructing a policy. Policies map states to *fine-grained actions*: e.g., when the arm is near the handle it moves slightly forward, and when the arm is holding the handle it pulls slightly backwards. In practice, this means the robot must reason about hundreds of low-level decisions throughout the task (i.e., each action of the policy). But at a high-level the drawer task can be broken down into three stages: reaching the handle, grasping the handle, and sliding the drawer open. Hence, the robot could solve this task — and maximize its given reward — by learning these *high-level waypoints* and interpolating between them.

In this paper we propose a waypoint-based approach for model-free reinforcement learning. Our work focuses on robot arms: inspired by recent research [1]–[3], we recognize

This work is supported in part by NSF Grants #2129201 and #2205241. The authors are with the Collaborative Robotics Lab (Collab), Dept. of Mechanical Engineering, Virginia Tech, Blacksburg, VA 24061. Corresponding author’s email: mehtashaunak@vt.edu

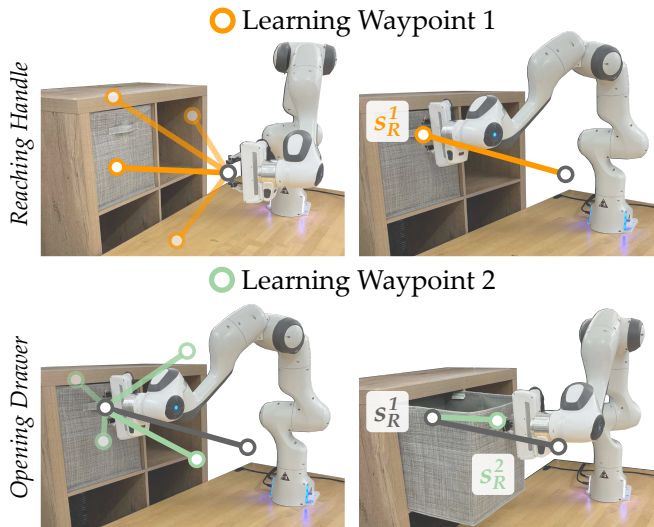


Fig. 1. Our waypoint-based approach for model-free reinforcement learning in manipulation tasks. The robot arm learns where to place the next waypoint to maximize its reward by solving a multi-armed bandit. We then freeze the learned models for waypoint i , and repeat the process for waypoint $i + 1$. This approach learns the desired task across a distribution of initial states; i.e., the location and angle of the drawers can change at each interaction.

that many robot manipulation tasks can be broken down into a sequence of waypoints. Moving between these waypoints is well understood: robot arms can leverage low-level controllers to track a reference trajectory [4], [5]. But to learn the correct waypoints in the first place, the robot must trade-off between exploring the workspace and exploiting high performing areas. Instead of formalizing this reinforcement learning problem as a Markov decision process — a standard approach in robotics [6] — our insight is that:

Each waypoint is a continuous multi-armed bandit problem, where the arm is the waypoint the robot selects and the reward for moving to that waypoint is unknown a priori.

Using this insight we introduce a method that builds trajectories *one waypoint at a time* to maximize the robot’s reward. When applied to Figure 1, our algorithm causes the robot to iteratively sample a waypoint and then roll-out a trajectory that terminates at that waypoint (e.g., trying a point to the left of the drawer). Based on the measured rewards for this rolled-out trajectory, the robot updates its estimate of the waypoint reward and optimizes the current waypoint (e.g., correctly reaching the drawer handle during the next roll-out). The robot then saves what it has learned about waypoint i and repeats this process for waypoint $i + 1$. As we will show in our experiments, this leads to robots that learn to open the drawer in fewer interactions than state-of-the-art baselines.

Overall, we make the following contributions:

Formulating as Sequential Multi-Armed Bandits. We use waypoints to write model-free reinforcement learning for robot manipulation tasks as a sequence of continuous multi-armed bandits, where each bandit problem corresponds to one waypoint along the robot’s learned trajectory. We theoretically demonstrate that this proposed formulation can have lower regret bounds than existing frameworks.

Learning Waypoints via Posterior Sampling. We next introduce an algorithm that approximately solves the sequence of multi-armed bandits through posterior sampling. Our method maintains an ensemble of models to estimate the reward function for a given waypoint; at each interaction the robot applies constrained optimization to select a waypoint that maximizes this estimated reward.

Testing in Simulated and Real Environments. We test our proposed approach across six simulated benchmark tasks, and two real-world robotic manipulation tasks. The results suggest that our method leads to higher rewards and faster convergence than SAC or PPO baselines.

II. RELATED WORK

We will introduce a reinforcement learning approach for robot manipulators [6] that is based on trajectory waypoints.

Hierarchical Reinforcement Learning. Similar to our approach, work on hierarchical reinforcement learning breaks down a robot’s behavior into high-level and low-level policies [7]. The low-level policies are temporally extended actions (e.g., subtasks, options, goals), and the high-level policy connects these subtasks to complete the overall task. For example, a robot arm can learn to insert a peg into a block by combining grasping and reaching subtasks [8]. The low-level policies can be parameterized (e.g., reaching for a given xyz position) [8], [9] or learned from scratch (e.g., policies conditioned on latent variables) [10], [11]. In either case, hierarchical reinforcement learning has the potential to accelerate exploration by reducing the number of actions the high-level policy must take to explore its workspace. Our approach can be viewed as an *instance* of hierarchical reinforcement learning with high-level waypoints and a low-level controller that guides the robot to each waypoint.

Motion Planning and Reinforcement Learning. Within our proposed method we learn a trajectory to maximize reward in model-free settings. Along these lines, recent research has used motion planning as a building block towards larger manipulation tasks [12]. In particular, we highlight existing methods that combine motion planning and reinforcement learning. In some of these works reinforcement learning is used to move to nearby goals, and a motion planner connects these low-level waypoints into a complete task plan [13], [14]. Alternatively, in other works motion planning finds *how* to reach a desired goal, and reinforcement learning determines *where* to place these intermediate goals for the larger task [15], [16]. Prior research suggests that this combination of motion planning and reinforcement learning is effective for long-horizon tasks where the robot arm must precisely

move to specific states (e.g., to pick up a block, the robot must first place its gripper directly above that block).

Posterior Sampling for Reinforcement Learning. Unlike these previous approaches, we will formulate the problem of learning the robot’s trajectory as a sequence of multi-armed bandits, where each bandit corresponds to a waypoint. One effective heuristic for solving multi-armed bandits is posterior sampling (i.e., Thompson sampling) [17]. Posterior sampling has previously been extended to reinforcement learning, with provable regret bounds in discrete state-action spaces [18]. However, this same approach is not tractable in continuous, high-dimensional environments. Instead, recent research has developed accurate approximations using neural networks that maintain an *ensemble* of learned models [19], [20]. We will similarly leverage an ensemble to approximate posterior sampling within our proposed algorithm.

III. PROBLEM FORMULATION

We consider settings in which a robot arm is given the reward function for a manipulation task, and the robot must learn to optimize this reward without relying on a model of the environment. Model-free reinforcement learning approaches can already be applied to these problems settings; however, current approaches often take thousands to millions of roll-outs to learn benchmark tasks [6], [8]. To learn these same tasks in fewer interactions, we will present a reinforcement learning approach where the robot arm moves between sequentially placed waypoints.

MDP. Reinforcement learning seeks to maximize the expected cumulative reward across a finite-horizon Markov Decision Process (MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, f, r, H, s^0 \rangle$. Here $s \in \mathcal{S}$ is the world state and $a \in \mathcal{A}$ is the robot arm’s action. Returning to our motivating example of opening a drawer, s includes the robot’s pose and the position and displacement of the drawer, and a is the robot’s end-effector velocity. At each timestep t the robot receives reward $r(s^t)$ and transitions between states based on the dynamics $s^{t+1} = f(s^t, a^t)$. These dynamics are *unknown*: the robot does not have access to a model of how its end-effector velocity will affect the displacement of the drawer or other environment variables. By contrast, we assume that the reward function $r : \mathcal{S} \rightarrow \mathbb{R}$ is *known*. For instance, the robot observes +1 reward at states where the drawer is open (i.e., the drawer’s displacement exceeds a threshold).

The robot repeatedly attempts to complete the same task in its environment and optimize the MDP \mathcal{M} . Each interaction (i.e., each episode) lasts for a total of H timesteps. Between interactions the world resets to the start state $s^0 \in \mathcal{S}$ (i.e., the drawer is closed and the robot returns to its home position). Let K be the total number of interactions, i.e., the total number of times the robot can attempt to perform the task.

Regret. During each interaction the system visits a trajectory of world states $\tau = \{s^0, \dots, s^H\}$, such that τ^k is the system trajectory for the k -th interaction ($k \leq K$). Remembering that $r(s)$ is the reward at a given state, we define $R(\tau) = \sum_{s \in \tau} r(s)$ as the total reward across trajectory τ .

Our objective is to identify trajectories that minimize regret. Consistent with prior works [21], we define the regret as:

$$REG(K) = \sum_{k=1}^K R(\tau^*) - R(\tau^k) \quad (1)$$

where τ^* is the best-case trajectory that maximizes the total reward (e.g., a trajectory that fully opens the drawer). Intuitively, minimizing regret means the robot learns actions to complete the desired task in as few interactions as possible. For model-free reinforcement learning to be practical on real-world robot arms, we need methods that quickly reduce the regret from Equation (1) within a few interactions K .

IV. REINFORCEMENT LEARNING WITH SEQUENTIAL WAYPOINTS

Our insight is that many manipulation tasks for robot arms can be broken down into a sequence of high-level waypoints that the arm must visit [1]–[3]. Consider our running example of opening the drawer: here the robot arm (1) moves its end-effector to the handle, (2) grasps the handle, and then (3) pulls the drawer open. Put another way, the robot arm completes the drawer task by following a trajectory between three high-level waypoints. We will assume that the robot can move between the given waypoints using an existing controller for trajectory tracking. Hence, our key challenge is determining *where to place each subsequent waypoint* so that the robot’s motion minimizes its regret. In Section IV-A we use our insight to reformulate reinforcement learning as a sequence of continuous multi-armed bandits. In Section IV-B we list the assumptions behind this formulation, and then in Section IV-C we explore the theoretical implications and derive lower bounds for regret. Finally, in Section IV-D we present our algorithm for learning where to place each subsequent waypoint along the robot’s trajectory.

A. Reformulation as a Sequence of Multi-Armed Bandits

Introducing high-level waypoints lets us present a different formulation of the standard reinforcement learning problem described in Section III. Under this new formulation the robot’s high-level action is the choice of where to place the next waypoint (i.e., the next state of the robot arm). As we will show, this leads to a sequence of continuous multi-armed bandits, where each bandit seeks to optimize the next waypoint along the robot’s trajectory.

From Actions to Trajectories. Let $s_{\mathcal{R}} \in \mathcal{S}_{\mathcal{R}}$ be the state of the robot arm. The robot’s state $s_{\mathcal{R}}$ is a subset of the world state s such that $\mathcal{S}_{\mathcal{R}} \subseteq \mathcal{S}$. For example, $s_{\mathcal{R}}$ could be the pose of the robot’s end-effector and gripper, while s could include $s_{\mathcal{R}}$ plus the position and displacement of the drawer. The robot arm has direct control over state $s_{\mathcal{R}}$. Put another way, the robot’s end-effector velocity a directly adjusts the position and orientation of its end-effector and gripper.

Under our proposed approach $s_{\mathcal{R}}$ is a *waypoint*, and the robot arm completes its task by moving through a *trajectory of waypoints*. Consider our running example of opening a drawer: the robot could perform this task by moving to a

state $s_{\mathcal{R}}^1$ where the robot grips the drawer handle, and then a state $s_{\mathcal{R}}^2$ where the robot arm moves away to pull the drawer open. We therefore define the robot’s reference trajectory ξ as a sequence of desired waypoints: $\xi = \{s_{\mathcal{R}}^0, \dots, s_{\mathcal{R}}^T\}$. The robot arm has a maximum of T waypoints in its trajectory. The choice of T is left to the designer; however, we constrain $T < H$ so that the number of waypoints is less than the total number of timesteps in one interaction. In what follows ξ_i denotes a snippet of the trajectory ξ with i waypoints ($i \leq T$). For example, ξ_1 is a trajectory snippet with just one waypoint $s_{\mathcal{R}}^1$, and ξ_2 is a trajectory snippet with two waypoints such that $\xi_2 = \{s_{\mathcal{R}}^1, s_{\mathcal{R}}^2\}$.

Overall, ξ defines a reference trajectory that we want the robot arm to follow. To track this trajectory we assume access to a low-level robot controller that takes actions a to move the robot arm along ξ . There are a variety of robot controllers for following reference trajectories [4], and our approach is not tied to any specific controller choice. In our experiments we use impedance control [5]. In practice, impedance control causes the robot arm to linearly interpolate between the waypoints $s_{\mathcal{R}}$ while remaining compliant if the robot comes into contact with objects in the environment.

From Unknown Dynamics to Unknown Rewards. Given that the robot arm has chosen to roll-out trajectory ξ , we next want to determine how effectively this trajectory will perform the desired task. As the robot tracks ξ its low-level controller outputs actions a (e.g., end-effector velocities, opening and closing the robot gripper). These actions change not only the robot’s state $s_{\mathcal{R}}$ but also the world state s . Returning to our example, by moving the robot’s end-effector away from the drawer while holding the handle, the robot modifies the displacement of the drawer. Accordingly, the complete trajectory of world states $\tau = \{s^0, \dots, s^H\}$ is a function of the robot’s trajectory ξ :

$$\tau = g(s^0, \xi) \quad (2)$$

Here g depends on both the unknown dynamics $f(s, a)$ and the robot’s low-level controller. The world starts in state s^0 and the robot takes low-level actions a to follow trajectory ξ . We overload this g notation for trajectory snippets: $g(s^0, \xi_i)$ outputs the sequence of world states s when the robot executes the trajectory snippet ξ_i .

The robot does not know how its own trajectory ξ will map to changes in the world state τ . From our example: if the robot plans to reach some waypoint $s_{\mathcal{R}}^1$ and then another waypoint $s_{\mathcal{R}}^2$, the robot does not know beforehand whether this motion will open the drawer, push against the drawer, or miss the drawer altogether. Without knowing the mapping g the robot cannot anticipate what effects its trajectory will have on the world state; however, the robot can measure the reward for trajectories it has previously executed. After the robot follows ξ and observes the sequence of resulting world states τ , the robot’s total reward is: $R(g(s^0, \xi)) = R(\tau)$. Similarly, after the robot executes a trajectory snippet ξ_i , its measured reward is $R(g(s^0, \xi_i))$. We will write these

trajectory rewards more compactly as:

$$R_{\theta}(s^0, \xi) = R(g(s^0, \xi)) \quad (3)$$

where $\theta = \{\theta_1, \theta_2, \dots, \theta_T\}$ is a set of *reward parameters* induced by the unknown mapping g . Here θ_1 parameterizes to the reward for moving from the start state to waypoint $s_{\mathcal{R}}^1$, and θ_i parameterizes the reward for moving from waypoint $i - 1$ to waypoint i . The robot *does not initially know these reward parameters*: returning to our example, at the first interaction the robot does not know how moving from $s_{\mathcal{R}}^{i-1}$ to $s_{\mathcal{R}}^i$ will affect the drawer’s displacement.

Multi-Armed Bandits. Using waypoints $s_{\mathcal{R}}$ and reward parameters θ , we can express our problem setting as a *sequence of continuous multi-armed bandits* (MAB). Under this formulation the robot learns how to construct ξ one waypoint at a time. Each MAB in the sequence corresponds to a different waypoint: in the first MAB the robot learns waypoint $s_{\mathcal{R}}^1$, and in i -th MAB the robot learns waypoint $s_{\mathcal{R}}^i$. More formally, each individual MAB \mathcal{B}_i is defined by:

- The continuous space of robot states $s_{\mathcal{R}} \in \mathcal{S}_{\mathcal{R}}$
- The continuous space of reward parameters $\theta_i \in \Theta_i$ for the transition from waypoint $i - 1$ to waypoint i
- A prior over the reward parameters such that $\theta_i \sim P_i(\cdot)$

The robot operates within every MAB for a fixed number of interactions and then transitions to the next MAB. Below we overview both stages of this process for learning the robot’s trajectory. We present a more detailed algorithm for solving these sequential MABs in Section IV-D.

Within MABs. Imagine the robot is operating within MAB \mathcal{B}_i (i.e., the robot is trying to determine where to place the i -th waypoint). During an interaction the robot selects and plays a bandit arm. More specifically, the robot selects the i -th waypoint $s_{\mathcal{R}}^i$ and executes the trajectory snippet ξ_i . After the trajectory is complete, the robot measures the resulting reward $R_{\theta_i}(s^0, \xi_i)$ parameterized by the unknown weights θ_i . This cycle repeats at each interaction — the robot tests a new choice of waypoint $s_{\mathcal{R}}^i$ and then observes the reward for the corresponding trajectory snippet ξ_i .

Between MABs. As the robot operates within MAB \mathcal{B}_i it learns what waypoint $s_{\mathcal{R}}^i$ maximizes its measured reward. We fix this waypoint in place when the robot transitions to the next MAB \mathcal{B}_{i+1} . More generally, we *freeze the strategy* the robot uses to choose $s_{\mathcal{R}}^i$ so that within MAB \mathcal{B}_{i+1} the robot only explores waypoint $s_{\mathcal{R}}^{i+1}$. To build trajectories with multiple waypoints the robot iterates through the previously learned strategies. Returning to our example: perhaps in \mathcal{B}_1 the robot learned to reach the drawer and in \mathcal{B}_2 the robot learned to grasp the handle. When the robot explores waypoint $s_{\mathcal{R}}^3$ in MAB \mathcal{B}_3 , trajectory $\xi = \{s_{\mathcal{R}}^1, s_{\mathcal{R}}^2, s_{\mathcal{R}}^3\}$ will first move to the drawer ($s_{\mathcal{R}}^1$) and grasp the handle ($s_{\mathcal{R}}^2$).

B. Assumptions

Before we analyze this proposed MAB formulation we want to clarify the assumptions behind our approach. First, we assume that the robot arm can complete the desired task using a sequence of T waypoints. This assumption is violated

when the designer chooses too few waypoints (i.e., the robot cannot open the drawer when $T = 1$), or when the robot is faced with a task that does not break down into waypoints (i.e., the robot arm balancing an inverted pendulum).

Second, we assume that the reward $r(s)$ is composed of multiple local maxima, and each maxima corresponds to a stage of the task that can be completed by a single waypoint. For example, the reward function for opening a drawer could include terms for distance from the handle, whether the handle is grasped, and the displacement of the drawer. This assumption is violated when the reward function does not encode a necessary subtask. For example, if the robot needs to approach the drawer handle from above to open it — but the reward only scores the distance to the center of the handle — the robot will incorrectly place a waypoint at the center of the handle (and not approach it from above).

These assumptions limit the types of tasks for which our sequential MAB formulation applies, and also place an increased emphasis on reward design. However — as we will show in our experiments — a variety of manipulation scenarios still satisfy the listed requirements. We also note that automated reward design is an ongoing research topic, and in future work methods such as [22] could be leveraged to mitigate our second assumption.

C. Lower Bounds on Regret

In Section IV-A we reformulated reinforcement learning with waypoints as a sequence of MABs. Here we explore the theoretical outcomes of this formulation. Specifically, we compare lower bounds on regret — as defined in Equation (1) — when using the standard MDP formulation and our special case MAB formulation. These lower bounds correspond to ideal performance: i.e., if we identify optimal algorithms for both of the problem formulations, how quickly and efficiently will the robot learn to complete the task? To take advantage of existing analysis, we consider settings where both the state space \mathcal{S} and the action space \mathcal{A} are *discrete*. In our experiments (Sections V and VI) we will test whether these theoretical results extend to continuous state-action spaces.

Discrete Notation. Let $|\mathcal{A}|$ be the number of actions, let $|\mathcal{S}|$ be the number of world states, and let $|\mathcal{S}_{\mathcal{R}}|$ be the number of robot states. As a reminder, H is the time horizon of one interaction and K is the total number of interactions. Within each MAB \mathcal{B}_i the robot can select any state from $\mathcal{S}_{\mathcal{R}}$. Hence, for one multi-armed bandit there are $|\mathcal{S}_{\mathcal{R}}|$ discrete arms. Because we fix the chosen arm in MAB \mathcal{B}_i when transitioning to MAB \mathcal{B}_{i+1} , across the sequence of T MABs the robot reasons over a total of $T \cdot |\mathcal{S}_{\mathcal{R}}|$ discrete arms.

Regret Bounds. Bubeck and Cesa-Bianchi [23] identified a lower bound on regret for any learning algorithm in an MAB. When applied to our sequential MAB from Section IV-A, this lower bound becomes: $REG(K) \geq \Omega(\sqrt{KT \cdot |\mathcal{S}_{\mathcal{R}}|})$.

Other research has identified lower bounds on the regret for any reinforcement learning algorithm that repeatedly interacts with a finite-horizon MDP [24]. When applied to our MDP \mathcal{M} from Section III, this lower bound becomes:

$REG(K) \geq \Omega(\sqrt{KH^2|\mathcal{S}||\mathcal{A}|})$. Osband and Van Roy conjecture that this lower bound is unimprovable [21].

These results quantify the lowest regret (i.e., the best performance) we could achieve under both problem formulations. Put another way, these bounds tell us about the minimum number of interactions the robot will need before it can complete the underlying task. It is not yet clear what algorithms will consistency reach these lower bounds. However, comparing the lower bounds does provide theoretical justification for when we should formulate the problem as an MDP \mathcal{M} (and solve using standard reinforcement learning methods) or as a sequence of MABs \mathcal{B} (and solve using our proposed approach). As $K \rightarrow \infty$, the lower bound for our MAB formulation is less than the MDP lower bound when:

$$T|\mathcal{S}_{\mathcal{R}}| < H^2|\mathcal{S}||\mathcal{A}| \quad (4)$$

We previously defined $T < H$ and $|\mathcal{S}_{\mathcal{R}}| \leq |\mathcal{S}|$. Hence, Equation (4) suggests that — in scenarios where our assumptions from Section IV-B apply — the ideal performance when learning one waypoint at a time is better than the ideal performance for learning a robot policy.

D. Approximate Solution with Posterior Sampling

In settings where the robot’s task can be broken down into a sequence of waypoints, our theoretical analysis suggests that the MAB formulation from Section IV-A is beneficial. Here we propose an approximate solution to this sequential MAB that constructs the robot’s trajectory one waypoint at a time. Our approximation is based on *posterior sampling*, a heuristic for multi-armed bandit problem [17]. The core idea in posterior sampling is to maintain a distribution over the unknown reward parameters θ ; at each iteration, the robot samples a value of θ from the posterior and then rolls-out the bandit arm that maximizes reward R_{θ} . Based on the measured reward from this roll-out, the robot updates its distribution over θ and then repeats the process using the new posterior. Unfortunately, we cannot directly apply posterior sampling because our state and parameter spaces are continuous and high-dimensional. Instead, our algorithm approximates the posterior distribution through an *ensemble* of reward models [19], [20]. Below we explain how we learn this ensemble within MAB \mathcal{B}_i , and also how we freeze the learned weights when transitioning to the next MAB \mathcal{B}_{i+1} .

Algorithm. See Algorithm 1 for our proposed approach. Our method is composed of two main parts to mirror the structure of sequential multi-armed bandits (Section IV-A). *Within* a given MAB the robot approximates posterior sampling through an ensemble of reward models, and selects the next waypoint to maximize the estimated reward. *Between* MABs the robot saves these trained reward models; during future MABs the robot can refer back to the saved models to determine the previous waypoints along its trajectory.

Within MABs. At the start of MAB \mathcal{B}_i the robot initializes an ensemble of N reward models (lines 4 – 6). These reward models estimate the unknown reward function $R_{\theta_i}(s^0, \xi_i)$ from Equation (3). This ensemble of reward

Algorithm 1 Learning Waypoints via Posterior Sampling

```

1: Initialize buffer of reward models  $\mathcal{R} = \{\}$ 
2: Initialize waypoint index  $i = 0$ 
3: while  $i < T$  do
4:    $i \leftarrow i + 1$  ▷ Transition to next waypoint
5:   Initialize  $N$  reward models with weights  $\theta_{i,1}, \dots, \theta_{i,N}$ 
6:   Initialize dataset  $\mathcal{D} \leftarrow \{\}$ 
7:   for each interaction in MAB  $\mathcal{B}_i$  do
8:     Measure world start state  $s^0$ 
9:     Initialize trajectory  $\xi \leftarrow \{\}$  ▷ Build trajectory
10:    for  $j = 1, \dots, i - 1$  do
11:       $s_{\mathcal{R}}^j \leftarrow \arg \max_{s_{\mathcal{R}}} \frac{1}{N} \sum_{n=1}^N R_{\theta_{j,n}}(s^0, (\xi, s_{\mathcal{R}}))$ 
12:       $\xi \leftarrow (\xi, s_{\mathcal{R}}^j)$  ▷ Add previous waypoints
13:    end for
14:    Sample one or more indices  $n \sim \text{uniform}(1, N)$ 
15:     $s_{\mathcal{R}}^i \leftarrow \arg \max_{s_{\mathcal{R}}} R_{\theta_{i,n}}(s^0, (\xi, s_{\mathcal{R}}))$ 
16:     $\xi \leftarrow (\xi, s_{\mathcal{R}}^i)$  ▷ Add  $i$ -th waypoint
17:    Roll-out trajectory  $\xi$  in MDP  $\mathcal{M}$ 
18:    Measure reward  $R$  for dataset  $\mathcal{D} \leftarrow (s^0, \xi, R)$ 
19:    for each reward model  $n$  do
20:       $\mathcal{L} \leftarrow \sum_{(s^0, \xi, R) \in \mathcal{D}} \|R_{\theta_{i,n}}(s^0, \xi) - R\|$ 
21:      Update weights  $\theta_{i,n}$  to minimize loss  $\mathcal{L}$ 
22:    end for
23:  end for
24:   $\mathcal{R} \leftarrow (\theta_{i,1}, \dots, \theta_{i,N})$  ▷ Save learned reward models
25: end while

```

models approximates the posterior distribution over θ_i . The robot samples from this ensemble to estimate the reward for the current interaction (line 14), and applies constrained optimization to identify the next waypoint $s_{\mathcal{R}}^i$ that maximizes the sampled reward (lines 15 – 16). The robot then rolls-out the resulting trajectory ξ in the environment using its low-level controller (lines 17 – 18). We record the initial world state, the robot’s trajectory of waypoints, and the measured reward (i.e., whether the drawer was opened during the interaction). In lines 19 – 21 the robot updates its posterior distribution over θ_i by training each reward model to match the measured rewards — over repeated interactions, the ensemble of reward models should converge towards the true reward function $R_{\theta_i}(s^0, \xi_i)$. This results in waypoints $s_{\mathcal{R}}^i$ which maximize the trajectory reward and complete the next stage of the task (i.e., reach for the handle or pull the drawer open). In practice, sampling from an ensemble of models causes the robot to trade-off between exploring new waypoints and exploiting the best performing waypoints.

Between MABs. The process we have described so far determines the i -th waypoint along the trajectory. But what about the previous waypoints $j = 1$ to $j = i - 1$? After completing MAB \mathcal{B}_{i-1} the robot saves the ensemble of reward models it has learned (line 24) When the robot moves on to the next MAB \mathcal{B}_i , we leverage these saved models at each interaction to find the previous waypoint $s_{\mathcal{R}}^{i-1}$. Looking specifically at lines 10 – 12, the robot loads each ensemble of models from its buffer (from $j = 1$ to $j = i - 1$), and

then applies constrained optimization to solve for waypoints $\xi = \{s_{\mathcal{R}}^1, \dots, s_{\mathcal{R}}^{i-1}\}$. We emphasize that the reward models used to select these previous waypoints are frozen; the robot does not continue to train or modify the previous models. Instead, the robot only focuses on the i -th waypoint while adding that new waypoint to a trajectory constructed across the sequence of previous MABs.

Implementation Details. Our experiments used an ensemble of $N = 10$ reward models. Each model was a fully connected multi-layer perceptron with two hidden layers and a leaky ReLU activation function. The reward models were updated by Adam with a learning rate of 0.001 and MSE loss. To find the waypoint that optimizes these reward models we applied constrained optimization — in our experiments, we leveraged Sequential Least Squares Programming (SLSQP). In practice this optimizer can get stuck in local maxima; to ameliorate this issue we used multiple initial seeds. A repository of our code is available here: <https://github.com/VT-Collab/rl-waypoints>

V. BENCHMARK SIMULATIONS

In this section we evaluate how our proposed approach learns new manipulation tasks in simulated environments. We performed these benchmark experiments in *robosuite*, a simulated robot environment with a set of standard manipulation tasks for robot arms [25]. Across these benchmark tasks we compared our proposed approach to state-of-the-art reinforcement learning algorithms, and measured the rewards (i.e., the task performance) achieved by each method.

Independent Variables. We compared the performance of our proposed Algorithm 1 (**Ours**) to four baselines. First, we implemented soft actor-critic (**SAC**) [26] and proximal policy optimization (**PPO**) [27]. Both **SAC** and **PPO** are model-free reinforcement learning algorithms that build a policy for selecting low-level robot actions a based on state s .

We also extended **SAC** and **PPO** to study their performance when using *high-level waypoints* instead of *low-level actions*. We will refer to these modified approaches as **SAC-wp** and **PPO-wp**. Similar to the original algorithms, both methods learn a policy that inputs world states s . But instead of outputting low-level actions a , now **SAC-wp** and **PPO-wp** output the next waypoint $s_{\mathcal{R}}^t$ that the robot should visit. We applied the same impedance controller as in **Ours** to move the robot arm between waypoints. The purpose of **SAC-wp** and **PPO-wp** was to test whether using high-level waypoints is the only advantage of our proposed approach: if **Ours** outperforms these baselines, this suggests that the framework we developed to learn the waypoints is also beneficial.

Environment. We used the benchmark tasks defined in *robosuite* [25] to evaluate the performance of each algorithm (see Figure 2). In *Lift* the robot needs to pick up a block. For the *Stack* task the robot has to pick up one block and stack it on top of another block. In *Nut Assembly* the robot picks up a nut and fits it on a peg. Next, in *Pick-and-Place* the robot picks up different objects in the environment and places them in their respective containers. These four tasks all focused on

the *xyz* actions of the robot’s end-effector. We also studied two manipulation tasks where the robot adjusts its position and orientation. In *Door-simple* the robot needs to reach for the door latch and pull the door open. In the more complex *Door-latch* task, the robot must turn the latch before it can open the door. We emphasize that the initial world state for each of these tasks was randomized at the start of every episode. For instance, the door was positioned at a different angle, and the blocks were placed at new locations.

Procedure. All the experiments were performed using a 7-DoF Franka-Emika robot arm with impedance control in end-effector space. Each task had an episode length of $H = 100$ timesteps. Across all environments **Ours**, **SAC-wp** and **PPO-wp** were trained to complete the task using $T = 2$ waypoints: the low-level controller used 50 timesteps to reach waypoint 1, and another 50 timesteps to reach waypoint 2. We made slight modifications to the given reward functions for the *Stack*, *Nut Assembly*, and *Pick-and-Place* tasks by increasing the rewards for completing each stage of the task and adding penalties for knocking the objects over.

Dependent Variables. The robot’s task performance was measured using reward. If the robot encountered a trajectory of world state τ during a given episode, we reported: $R(\tau) = \sum_{s \in \tau} r(s)$. Higher rewards indicate better performance.

Results. We separated our results into two parts (see Figure 2 and Table I). Figure 2 shows the rewards achieved by each model-free reinforcement learning algorithm as it trained in the environment. Every interaction (i.e., every episode) lasted $H = 100$ timesteps, and we reported the total reward across that episode. After the robot completed K total interactions, we then saved the learned models and tested their performance. Our results from these separate tests are listed in Table I. Here the simulated robot attempted to complete each manipulation task 100 times using the models it had learned from training. Similar to training, during evaluation the position and orientation of objects in the environment was randomized at the start of every interaction.

Overall, none of the baselines were able to learn the tasks correctly within the limited number of episodes available for training. Across all tasks, we observed that **SAC** learned to reach and sometimes grasp the desired objects, while **PPO** only learned to reach for these objects. We also noticed that **SAC-wp** performed well at the start of training, but its performance dropped as the training progressed. We hypothesize that this may have occurred because — instead of learning one waypoint at a time — **SAC-wp** was simultaneously learning and adjusting both waypoints of the task. This may have prevented the robot from determining which waypoints were leading to higher rewards: e.g., did the robot succeed because of the position of waypoint 1 or waypoint 2?

Across both training and evaluation **Ours** outperformed the baselines. In Figure 2 we highlight that **Ours** had sudden jumps in episode reward. These jumps corresponded to episodes where the robot *added a new waypoint* to its trajectory. For example, in *Lift* the robot arm solved the multi-armed bandit for waypoint 1 during the first 200 episodes,

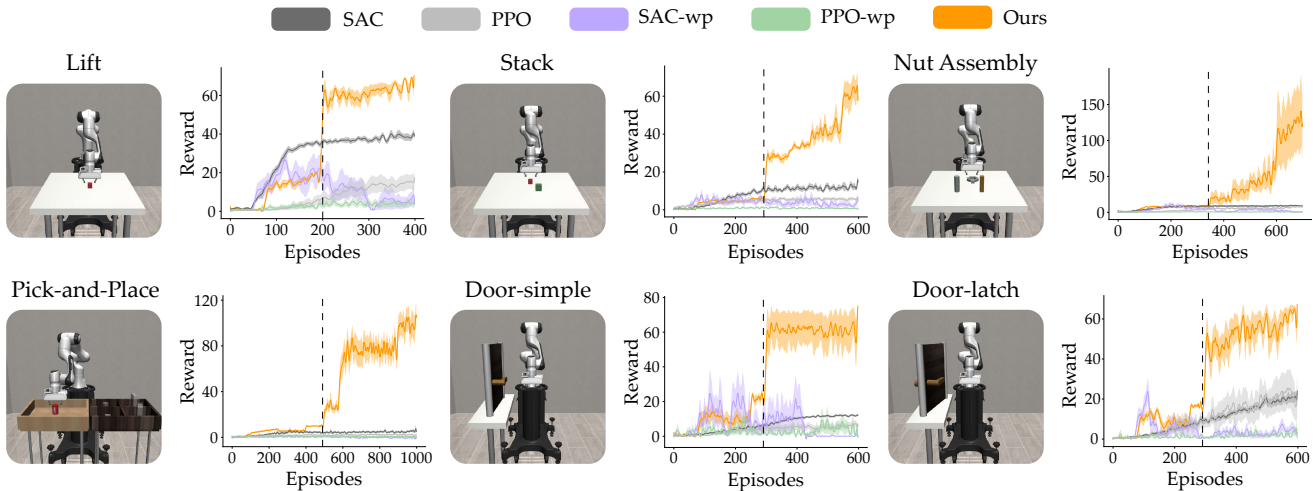


Fig. 2. Simulation environments and rewards for six manipulation tasks. These benchmark tasks are taken from robosuite [25]. For each task the image on the left shows the environment setup, and the plot on the right shows the robot’s rewards averaged over five runs. Higher rewards indicate better task performance. The dashed lines in the reward plots correspond to the episodes where **Ours** approach added a new waypoint to the trajectory. Adding these new waypoints often causes a sharp increase in the robot’s reward. This sudden change occurs because the new waypoint enables the robot to complete the next part of the task: e.g., in *Lift* the robot needs one waypoint to grasp the block and a second waypoint to lift the block.

TABLE I

EVALUATION RESULTS. AFTER TRAINING IN FIGURE 2, WE SAVED THE LEARNED MODELS AND EVALUATED THEIR PERFORMANCE ACROSS 100 EPISODES. WE REPORT AVERAGE REWARD \pm STANDARD ERROR.

Task	Mehtod				
	SAC	PPO	SAC-wp	PPO-wp	Ours
Lift	44.31 \pm 2.06	17.58 \pm 0.34	0.46 \pm 0.02	0.22 \pm 0.06	68.83 \pm 0.02
Stack	12.20 \pm 0.42	3.04 \pm 0.19	0.01 \pm 0.0	0.86 \pm 0.09	70.37 \pm 3.56
Nut Assembly	9.51 \pm 0.14	8.54 \pm 0.33	0.10 \pm 0.01	0.74 \pm 0.09	190.14 \pm 11.7
Pick-and-Place	6.69 \pm 0.24	1.37 \pm 0.10	1.11 \pm 0.17	0.24 \pm 0.05	89.24 \pm 2.87
Door-simple	9.78 \pm 0.23	4.58 \pm 0.20	0.01 \pm 0	1.67 \pm 2.59	63.01 \pm 2.19
Door-latch	15.89 \pm 0.17	6.64 \pm 0.26	0.14 \pm 0.01	0.29 \pm 0.03	48.70 \pm 2.58

and then transitioned to the next MAB starting at episode 201. The reward increased at this transition because the robot progressed to the next stage of the task. Returning to *Lift*, in the first 200 episodes the robot learned one waypoint to reach and grasp the randomly initialized block. Starting from episode 201, the robot applied what it had learned to grasp the block, and then explored its second waypoint to decide how to carry that block. In summary, these simulation results suggest that **Ours** can efficiently learn a range of manipulation tasks in a limited number of interactions by breaking the task down into several waypoints, and learning the trajectory one waypoint at a time.

VI. REAL-WORLD EXPERIMENTS

In this section we explore whether our proposed algorithm enables robot arms to learn manipulation tasks from scratch in real-world environments. We compare our approach (**Ours**) with the best-performing baseline from our simulations (**SAC**) across two different manipulation tasks. Within real-world environments with an actual robot arm, we find that Algorithm 1 leads to shorter training periods and more accurate task performance. See the robot’s learned behaviors here: <https://youtu.be/MMEd-1Yfq4Y>

Experimental Setup. We conducted our real-world experiments on a 7-DoF Franka Emika robot arm (see Figure 3).

We trained the robot to perform two different manipulation tasks: lifting an object (*Lift*) and opening a drawer (*Drawer*). The robot used **Ours** and **SAC** to attempt to learn each task.

For both tasks the episodes lasted $H = 140$ timesteps. The robot used end-effector velocity control to take actions within the environment. Under **SAC** the robot moved its end-effector and gripper based on the action output by the learned policy. With **Ours**, the robot learned two waypoints (70 timesteps each), and leveraged the same low-level controller to interpolate between its start state and the waypoints along its reference trajectory $\xi_{\mathcal{R}}$.

At the beginning of each episode the robot’s initial position was constant, but the location of the object or drawer was uniformly randomized within a 6-by-6 cm plane. We selected this smaller range to minimize the number of training episodes and prevent the robot from running into its joint limits. We also applied boundary conditions to stop the robot from colliding with the table or attempting to move beyond its workspace. For *Lift*, we used the same reward function as in robosuite’s lift task. For *Drawer*, we defined a dense reward function that encouraged the robot to grasp the handle and then pull the drawer out in a straight line.

Results. We first measured the episode rewards throughout training. Figure 3 shows the training rewards averaged across three runs for both tasks and methods. The maximum reward the robot achieved with **Ours** was about 1.3 times **SAC** in *Lift*, and about 7.75 times **SAC** in *Drawer*.

After training completed, we used the robot’s learned models to evaluate the robot’s final success rate for both tasks. A robot trained with **Ours** grasped and lifted the randomly placed block in 16/20 evaluation trials (80% success rate). Similarly, **Ours** robot was able to grasp and at least partially open the drawer in 20/20 evaluation trials. By contrast, under **SAC** the robot was never able to complete either task given the limited training budget.

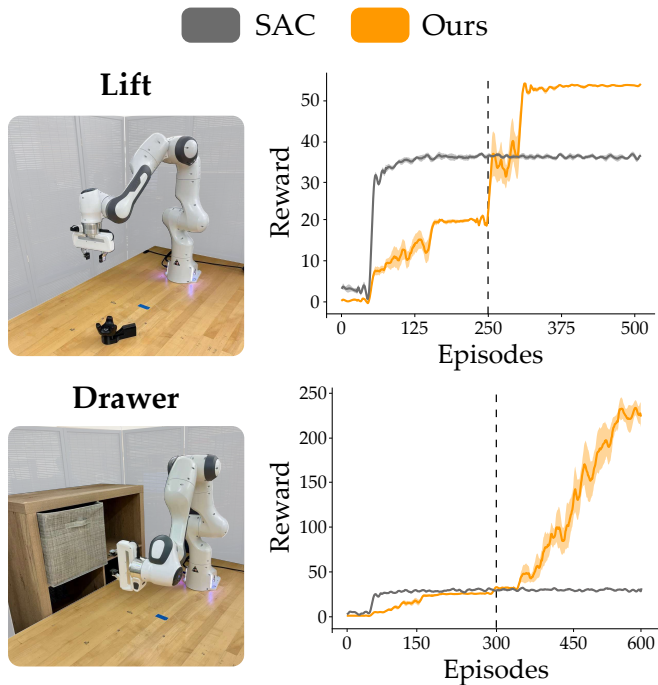


Fig. 3. Setup and results from our real-world experiments in Section VI. In *Lift* the robot learned to pick up an item, and in *Drawer* it learned to open a drawer (also see Figure 1). The robot measured the initial position of the item or drawer as a part of the start state s^0 . SAC learned a policy that often moved to the object of interest, but did not correctly interact with that object. When using **Ours**, the robot built a trajectory of two waypoints: the first waypoint grasped the object, and then the second waypoint interacted with that object (e.g., picked the item up, pulled the drawer open). The dashed line in the reward plots corresponds to the episode where **Our** robot added a second waypoint to its trajectory.

VII. CONCLUSION

In this paper we focused on model-free reinforcement learning for robot arms. We recognized that many of the everyday manipulation tasks that we want robots to learn can be broken down into a series of high-level waypoints (e.g., reaching an object). We therefore proposed a framework for waypoint-based reinforcement learning, where the robot learns new tasks by building a trajectory one waypoint at a time. Our key contribution was reformulating this problem as a sequence of multi-armed bandits: our theoretical analysis suggests that best-case solutions to this bandit formulation will outperform standard approaches. We next introduced one possible algorithm for solving the sequential bandit problem. Our proposed approach leveraged an ensemble of models to approximate posterior sampling: in practice, this method learned each subsequent waypoint to greedily maximize the robot’s reward. We found that our approach outperformed commonly used model-free reinforcement learning algorithms across a set of simulated and real-world manipulation tasks. In future works we will explore more complicated tasks that require a higher number of waypoints.

REFERENCES

[1] S. Belkhal, Y. Cui, and D. Sadigh, “HYDRA: Hybrid robot actions for imitation learning,” in *Conference on Robot Learning*, 2023.

[2] B. Akgun, M. Cakmak, K. Jiang, and A. L. Thomaz, “Keyframe-based learning from demonstration: Method and evaluation,” *International Journal of Social Robotics*, vol. 4, pp. 343–355, 2012.

[3] L. X. Shi, A. Sharma, T. Z. Zhao, and C. Finn, “Waypoint-based imitation learning for robotic manipulation,” in *Conference on Robot Learning*, 2023.

[4] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. John Wiley & Sons, 2020.

[5] S. Haddadin and E. Croft, “Physical human–robot interaction,” in *Springer Handbook of Robotics*. Springer, 2016.

[6] D. Han, B. Mulyana, V. Stankovic, and S. Cheng, “A survey on deep reinforcement learning algorithms for robotic manipulation,” *Sensors*, vol. 23, no. 7, 2023.

[7] R. S. Sutton, D. Precup, and S. Singh, “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning,” *Artificial Intelligence*, pp. 181–211, 1999.

[8] S. Nasiriany, H. Liu, and Y. Zhu, “Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks,” in *IEEE International Conference on Robotics and Automation*, 2022.

[9] O. Nachum, S. S. Gu, H. Lee, and S. Levine, “Data-efficient hierarchical reinforcement learning,” in *Advances in Neural Information Processing Systems*, 2018.

[10] P.-L. Bacon, J. Harb, and D. Precup, “The option-critic architecture,” in *AAAI*, 2017.

[11] J. Zhang, H. Yu, and W. Xu, “Hierarchical reinforcement learning by discovering intrinsic options,” in *International Conference on Learning Representations*, 2021.

[12] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, “Integrated task and motion planning,” *Annual Review of Control, Robotics, and Autonomous Systems*, 2021.

[13] B. Eysenbach, R. R. Salakhutdinov, and S. Levine, “Search on the replay buffer: Bridging planning and reinforcement learning,” in *Advances in Neural Information Processing Systems*, 2019.

[14] R. Gieselmann and F. T. Pokorny, “Planning-augmented hierarchical reinforcement learning,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5097–5104, 2021.

[15] F. Xia, C. Li, R. Martín-Martín, O. Litany, A. Toshev, and S. Savarese, “Relmogen: Integrating motion generation in reinforcement learning for mobile manipulation,” in *IEEE International Conference on Robotics and Automation*, 2021.

[16] J. Yamada, Y. Lee, G. Salhotra, K. Pertsch, M. Pflueger, G. Sukhatme, J. Lim, and P. Englert, “Motion planner augmented reinforcement learning for robot manipulation in obstructed environments,” in *Conference on Robot Learning*, 2021.

[17] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, and Z. Wen, “A tutorial on thompson sampling,” *Foundations and Trends in Machine Learning*, vol. 11, no. 1, pp. 1–96, 2018.

[18] I. Osband and B. Van Roy, “Why is posterior sampling better than optimism for reinforcement learning?” in *International Conference on Machine Learning*, 2017.

[19] C. Qin, Z. Wen, X. Lu, and B. Van Roy, “An analysis of ensemble sampling,” *Advances in Neural Information Processing Systems*, 2022.

[20] K. Lee, M. Laskin, A. Srinivas, and P. Abbeel, “Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning,” in *International Conference on Machine Learning*, 2021.

[21] I. Osband and B. Van Roy, “On lower bounds for regret in reinforcement learning,” *arXiv preprint arXiv:1608.02732*, 2016.

[22] F. Memarian, W. Goo, R. Lioutikov, S. Niekum, and U. Topcu, “Self-supervised online reward shaping in sparse-reward environments,” in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2021.

[23] S. Bubeck and N. Cesa-Bianchi, “Regret analysis of stochastic and nonstochastic multi-armed bandit problems,” *Foundations and Trends in Machine Learning*, vol. 5, no. 1, pp. 1–122, 2012.

[24] O. D. Domingues, P. Ménard, E. Kaufmann, and M. Valko, “Episodic reinforcement learning in finite mdp: Minimax lower bounds revisited,” in *Algorithmic Learning Theory*, 2021.

[25] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu, “robosuite: A modular simulation framework and benchmark for robot learning,” in *arXiv:2009.12293*, 2020.

[26] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International Conference on Machine Learning*, 2018.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv:1707.06347*, 2017.