C-PLANNING: AN AUTOMATIC CURRICULUM FOR LEARNING GOAL-REACHING TASKS

Tianjun Zhang[*]
UC Berkeley
tianjunz@berkeley.edu

Benjamin Eysenbach[*]
Carnegie Mellon University
beysenba@cs.cmu.edu

Ruslan Salakhutdinov
Carnegie Mellon University

Sergey Levine
UC Berkeley

Joseph E. Gonzalez
UC Berkeley

ABSTRACT

Goal-conditioned reinforcement learning (RL) can solve tasks in a wide range of domains, including navigation and manipulation, but learning to reach distant goals remains a central challenge to the field. Learning to reach such goals is particularly hard without any offline data, expert demonstrations, and reward shaping. In this paper, we propose an algorithm to solve the distant goal-reaching task by using search at training time to automatically generate a curriculum of intermediate states. Our algorithm, Classifier-Planning (C-Planning), frames the learning of the goal-conditioned policies as expectation maximization: the E-step corresponds to planning an optimal sequence of waypoints using graph search, while the M-step aims to learn a goal-conditioned policy to reach those waypoints. Unlike prior methods that combine goal-conditioned RL with graph search, ours performs search only during training and not testing, significantly decreasing the compute costs of deploying the learned policy. Empirically, we demonstrate that our method is more sample efficient that prior methods. Moreover, it is able to solve very long horizons manipulation and navigation tasks, tasks that prior goal-conditioned methods and methods based on graph search fail to solve.[1]

1 INTRODUCTION

Whereas typical RL methods maximize the accumulated reward, goal-conditioned RL methods learn to reach any goal. Arguably, many tasks are more easily defined as goal-reaching problems than as reward maximizing problems. While many goal-conditioned RL algorithms learn policies that can reach nearby goals, learning to reach distant goals remains a challenging problem. Some prior methods approach this problem by performing search or optimization over subgoals at test time. However, these test-time planning methods either rely on graph search [7, 31], which scales poorly with dimensionality [14], or continuous optimization over subgoals [26], which is expensive and can result in model exploitation.

In this paper, we take a different tack and instead use search at training time to automatically generate a curriculum. When training the agent to reach one goal, our method first determines some intermediate waypoints enroute to that goal. Then, it commands the agent to reach those waypoints before navigating to the final destination. Collecting data in this manner improves the quality of data, allowing the agent to learn to reach distant goals. Importantly, our method does not perform planning at test time, decreasing the computational demands for deploying the learned agent.

Our curriculum does not require manual engineering or prior knowledge of the tasks. Rather, the curriculum emerges automatically when we use expectation maximization (EM) to maximize a lower bound on the probability of reaching the goal. The M-step corresponds to a prior method for goal-conditioned RL, while the E-step corresponds to graph search. Thus, EM presents a marriage between previous goal-conditioned RL algorithms and graph-search methods.

[*]Equal contribution.
[1]Code and videos of our results: https://ben-eysenbach.github.io/c-planning/

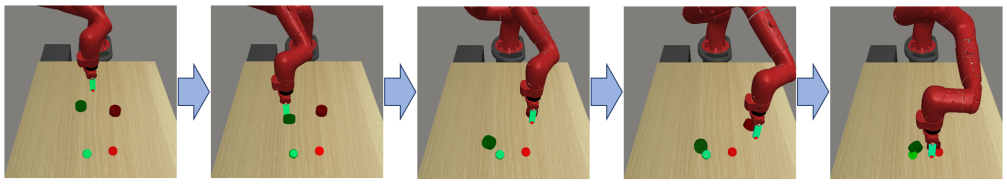
Figure 1: **C-Planning** is an algorithm for goal-conditioned RL that uses an automatic curriculum of waypoints to learn policies that can solve complex tasks. In this manipulation task, the goal requires moving the green puck to the green dot and the red puck to the red dot. Our method learns to solve this task, manipulating multiple objects in sequence, without requiring any reward functions, manual distance functions, or human demonstrations.

The main contribution of this work is a goal-conditioned RL algorithm, C-Planning, that excels at reaching distant goals. Our method uses an automatic curriculum of subgoals to accelerate training. We show that our curriculum emerges from applying variational inference to the goal-reaching problem. Unlike prior work, our method does not require graph search or optimization at test-time. Empirically, C-Planning not only matches but surpasses the performance of these prior search-based methods, suggesting that it is not just amortizing the cost of graph search. We empirically evaluate C-Planning on temporally-extended 2D navigation tasks and complex 18D robotic manipulation tasks. C-Planning improves the sample efficiency of prior goal-conditioned RL algorithms and manages to solve more difficult manipulations tasks such as rearranging multiple objects in sequence (see Fig. 1.). To the best of our knowledge, no prior method has learned tasks of such difficulty without requiring additional assumptions.

2 Related Work

The problem of learning to achieve goals has a long history, both in the control community Lyapunov [22] and the RL community [16]. Many prior papers approach goal-conditioned RL as a reward-driven, multi-task learning problem, assuming access to a goal-conditioned reward function. One unique feature of goal-conditioned RL, compared to other multi-task RL settings, is that it can also be approached using reward-free methods, such as goal-conditioned behavior cloning [5, 12, 23, 31] and RL methods that employ hindsight relabeling [8, 16, 21, 32]. While goal-conditioned behavior cloning methods are simple to implement and have shown excellent results on a number of real-world settings [24, 33], they are not guaranteed to recover the optimal policy without additional assumptions (e.g., determinism, online data collection). While both methods excel at certain control tasks, both often struggle to solve tasks with longer horizons.

To solve longer-horizon tasks, prior work has combined goal-conditioned RL with graph search, noting that goal-conditioned value functions can be interpreted as dynamical distances [6, 15, 26, 31]. These prior methods typically proceed in two phases: the first phase learns a goal-conditioned policy, and the second phase combines that policy with graph search. While these methods have demonstrated excellent results on a number of challenging control tasks, including real-world robotic navigation [24, 33], the stage-wise approach has a few limitations. First, the post-hoc use of graph search means that graph search cannot improve the underlying goal-conditioned policy. It is well known that the performance of RL algorithms is highly dependent on the quality of the collected data [17, 18, 20]. By integrating search into the learning of the underlying goal-conditioned policy, our method will improve the quality of the data used to train that policy. A second limitation of these prior methods is the cost of deployment: choosing actions using graph search requires at least $\mathcal{O}(|\mathcal{V}|)$ queries to a neural network.[2] In contrast, our method performs search at training time rather than test time, so the cost of deployment is $\mathcal{O}(1)$. While this design decision does increase the computational complexity of training, it significantly decreases the latency at deployment. Our method is similar to RIS [3], which also performs search during training instead of deployment. Our method differs from RIS in how search is used: whereas RIS modify the *objective*, our method uses search to modify the *data*. This difference not only unifies the objectives for RL and graph search, but also significantly improves the performance of the algorithm. , a subtle distinction that avoids favoring the learned policy and unifies the objectives for RL and graph search. We demonstrate the importance of this difference in our experiments.

[2]While computing all pairwise distances requires $\mathcal{O}(|\mathcal{V}|^2)$ time, only $|\mathcal{V}|$ edges change at each time step. The cost of computing the remaining edges can be amortized across time.

Effectively solving goal-conditioned RL problems requires performing good exploration. In the goal-conditioned setting, the quality of exploration depends on how goals are sampled, a problem studied in many prior methods [6, 10, 11, 27, 28, 30, 38, 39]. These methods craft objectives that try to optimize for learning progress, and the resulting algorithms achieve good results across a range of environments. Our method differs from these methods in that the method for optimizing the goal-conditioned policy and the method for sampling waypoints are jointly optimized using the same objective. We demonstrate the importance of this difference in our experiments.

Our method builds on the idea that reinforcement learning can be cast as inference problems [2, 19, 29, 35, 36, 40]. The observation that variational inference for certain problems corresponds to graph search is closely related to Attias [2]. Our work extends this inferential perspective to hierarchical models.

3 Preliminaries

We introduce the goal-conditioned RL problem and a recent goal-conditioned RL algorithm, C-learning, upon which our method builds.

Goal-Conditioned RL. We focus on controlled Markov processes defined by states $\mathbf{s}_t \in \mathcal{S}$ and actions $\mathbf{a}_t$. The initial state is sampled $\mathbf{s}_0 \sim p_0(\mathbf{s}_0)$ and subsequent states are sampled $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$. Our aim is to navigate to the goal states, $\mathbf{s}_g \in \mathcal{S}$, which are sampled from $\mathbf{s}_g \sim p_g(\mathbf{s}_g)$. We define a policy $\pi(\mathbf{a}_t \mid \mathbf{s}_t, \mathbf{s}_g)$ conditioned on both the current state and the goal. The objective is to maximize the probability (density) of reaching the goal in the future.

To derive our method, or any other goal-conditioned RL algorithm, we must make a modeling assumption about *when* we would like the agent to reach the goal. This modeling assumption is only used to derive the algorithm, not for evaluation. Formally, let $\Delta \in \mathbb{N}^+$ be a random integer indicating when we reach the goal. The user specifies a prior $p(\Delta)$. Most prior work (implicitly) uses the geometric distribution, $p(\Delta) = \text{GEOM}(1-\gamma)$. The geometric distribution is ubiquitous because it is easy to incorporate into temporal difference learning, not because it is a particularly reasonable prior belief on when the agent will solve the task or when the episode will terminate. Prior work has also considered non-geometric distributions [9]. In this paper, we use the negative binomial distribution, $p(\Delta) = \text{NEGBINOM}(p = 1-\gamma, n = 2)$. Given a prior $p(\Delta)$, we define the distribution over future states as

$$p_{p(\Delta)}^{\pi(\cdot|\cdot,\mathbf{s}_g)}(\mathbf{s}_{t+} = \mathbf{s}_{t+} \mid \mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{p(\Delta)}\left[p_{\Delta}^{\pi(\cdot|\cdot,\mathbf{s}_g)}(\mathbf{s}_{t+\Delta} = \mathbf{s}_{t+} \mid \mathbf{s}_t, \mathbf{a}_t)\right], \tag{1}$$

where $p_{\Delta}^{\pi}(\mathbf{s}_{t+\Delta} \mid \mathbf{s}_t, \mathbf{a}_t)$ is the probability density of reaching state $\mathbf{s}_{t+\Delta}$ exactly Δ steps in the future when sampling actions from $\pi(\mathbf{a}_t \mid \mathbf{s}_t, \mathbf{s}_g)$. For example, the γ-discounted state occupancy measure [13, 25] can be written as $p_{\text{GEOM}(1-\gamma)}^{\pi(\cdot|\cdot,\mathbf{s}_g)}$. Our objective for goal-reaching is to maximize the probability of reaching the desired goal:

$$\max_{\pi} \mathbb{E}_{p_g(\mathbf{s}_g)}\left[p_{\text{NEGBINOM}(p=1-\gamma,n=2)}^{\pi(\cdot|\cdot,\mathbf{s}_g)}(\mathbf{s}_{t+} = \mathbf{s}_g)\right]. \tag{2}$$

C-Learning. Our method builds upon a prior method for goal-conditioned RL, C-Learning [8]. C-Learning learns a classifier for predicting whether a state $\mathbf{s}_t$ comes from a future state density $p_{p(\Delta)}^{\pi(\cdot|\cdot,\mathbf{s}_g)}(\mathbf{s}_{t+} = \mathbf{s}_{t+} \mid \mathbf{s}_t, \mathbf{a}_t)$ or a marginal state density:

$$p_{p(\Delta)}^{\pi(\cdot|\cdot,\mathbf{s}_g)}(\mathbf{s}_{t+} = \mathbf{s}_{t+}) = \int p_{p(\Delta)}^{\pi(\cdot|\cdot,\mathbf{s}_g)}(\mathbf{s}_{t+} = \mathbf{s}_{t+} \mid \mathbf{s}_t, \mathbf{a}_t)\, p(\mathbf{s}_t, a_t)\, d\mathbf{s}_t\, d\mathbf{a}_t.$$

The Bayes-optimal classifier can be written in terms of these two distributions:

$$C_\theta(\mathbf{s}, \mathbf{s}_g) = \frac{p_{\text{GEOM}}^{\pi(\cdot|\cdot,\mathbf{s}_g)}(\mathbf{s}_g \mid \mathbf{s})}{p_{\text{GEOM}}^{\pi(\cdot|\cdot,\mathbf{s}_g)}(\mathbf{s}_g \mid \mathbf{s}) + p_{\text{GEOM}}^{\pi(\cdot|\cdot,\mathbf{s}_g)}(\mathbf{s}_g)}. \tag{3}$$

In C-Learning, this classifier acts as a value function for training the policy. Our method will use this classifier not only to update the policy, but also to sample waypoints.

4 C-Planning: Goal-Conditioned RL with Planning

We now present our method, C-Planning. The main idea behind our method is to decompose the objective of Eq. 2 into a sequence of easier goal-reaching problems. Intuitively, if we want a sub-optimal agent to navigate from some initial state s_0 to a distant goal state s_g, having the agent try to reach that state again and again will likely prove futile. Instead, we can command a sequence of goals, like a trail of breadcrumbs leading to the goal s_g. To select these waypoints, we will command the policy to reach those states it would visit *if* it were successful at reaching s_g. Sampling subgoals in this way can ease the difficulty of training the agent (see Fig. 2.).

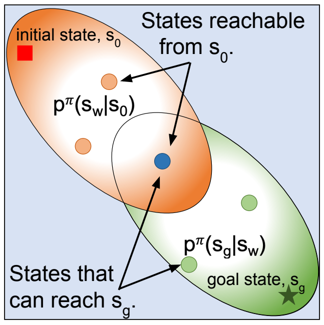

Figure 2: **Waypoint Sampling**: Our method samples waypoints that are reachable from the initial state, and from which the agent can reach the goal.

At a high-level algorithm summary, our method consists of two steps: using waypoing sampling to collect experience and performing standard goal-conditioned RL using the collected experience. The main challenge is the first step, as modeling the distribution over waypoints is difficult. In Sec. 4.1, we derive the optimal waypoint distribution, and then propose a simple practical method to sample from this distribution in Sec. 4.2.

4.1 Planning and Variational Inference

To formally derive our method for waypoint sampling, we cast the problem of goal-reaching as a latent variable problem. We assume that the agent starts at state s_0 and ends at state s_g, but the intermediate states that the agent will visit are unknown latent variables. This problem resembles standard latent variable modeling problems (e.g., VAE), where the intermediate state serves the role of the inferred representation. We can thus derive an evidence lower bound on the likelihood of reaching the desired goal:

Lemma 1. *The objective $\mathcal{L}$, defined below, is a lower bound on the goal-reaching objective (Eq. 2):*

$$\log p^{\pi(\cdot|\cdot,s_g)}_{\text{NEGBINOM}}(s_{t+} = s_g \mid s_0) \geq \mathbb{E}_{q(s_w|s_g,s_0)}\Big[\log p^{\pi(\cdot|\cdot,s_g)}_{\text{GEOM}}(s_{t+} = s_g \mid s_w) + \log p^{\pi(\cdot|\cdot,s_g)}_{\text{GEOM}}(s_w \mid s_0) - \log q(s_w \mid s_g, s_0)\Big] \triangleq \mathcal{L}(\pi, q(s_w \mid s_g, s_0)).$$

See Appendix A.1 for the proof. The lower bound, $\mathcal{L}$, depends on two quantities: the goal-conditioned policy and an *inferred* distribution over waypoints, $q(s_w \mid s_0, s_g)$. This bound holds for any choice of $q(s_w \mid s_0, s_g)$, and we can optimize this lower bound with respect to this waypoint distribution. We will perform expectation maximization [4] to optimize the lower bound, alternating between computing the optimal waypoint distribution and optimizing the goal-conditioned policy using this waypoint distribution. We now describe these two steps in detail.

E-Step. The E-step estimates the waypoint distribution, $q(s_w \mid s_0, s_g)$. The three terms in the lower bound indicate that the sampled waypoint should be reachable from the initial state, the goal state should be reachable from the sampled waypoint, and that the waypoint distribution should have high entropy. These first two terms resemble shortest-path planning, where distances are measured using log probabilities. Importantly, reachability is defined in terms of the capabilities of the current policy. We can analytically solve for the optimal waypoint distribution:

Lemma 2. *The optimal choice for the waypoint distribution satisfies:*

$$q^*(s_w \mid s_g, s_0) = \frac{p^{\pi(\cdot|\cdot,s_g)}_{\text{GEOM}}(s_g \mid s_w) p^{\pi(\cdot|\cdot,s_w)}_{\text{GEOM}}(s_w \mid s_0)}{\int p^{\pi(\cdot|\cdot,s_g)}_{\text{GEOM}}(s_g \mid s'_w) p^{\pi(\cdot|\cdot,s_w)}_{\text{GEOM}}(s'_w \mid s_0) ds'_w}.$$

See Appendix A.1 for a full derivation. We illustrate this waypoint distribution in Fig. 2. In general, accurately estimating the optimal distribution, q^*, is challenging. However, in the next section, we develop a simple algorithm that only requires learning a classifier instead of a generative model.

M-step. The M-step optimizes the lower bound with respect to the goal-conditioned policy. We can ignore the $-\log q(s_w \mid s_0, s_g)$ term, which does not depend on the policy. The remaining two terms look like goal-conditioned RL objectives, with a subtle but important difference in how the trajectories are sampled. When collecting data, a standard goal-conditioned RL algorithm initially samples a goal and then collects a trajectory where the policy attempts to reach that goal. Our method collects data in a different manner. The agent samples a goal, *then samples an intermediate waypoint that should lead to that goal.* The agent attempts to reach the waypoint *before* attempting to reach the goal. After the trajectory has been collected, the policy updates are the same as a standard goal-conditioned RL algorithm.

Two mental models. The algorithm sketch we have presented, which will be fleshed out in the subsequent section, can be interpreted in two ways. One interpretation is that the method performs a soft version of graph search during exploration, using a distance function of $d(\mathbf{s}_1, \mathbf{s}_2) = \log p_{\text{GEOM}}^{\pi(\cdot|\cdot,\mathbf{s}_2)}(\mathbf{s}_{t+} = \mathbf{s}_2 \mid \mathbf{s}_1)$. From this perspective, the method is similar to prior work that performs graph search during test-time [7, 31]. However, our method will perform search at training time. Extensive ablations in Sec. 5 show that our method outperforms alternative approaches that only perform search during test-time.

The second interpretation focuses on the task of reaching the goal from different initial states. The choice of waypoint distribution, and the policy's success at reaching those waypoints, determines the initial state distribution for this task. Intuitively, the best initial state distribution is one that sometimes starts the agent at the true initial state $\mathbf{s}_0$, sometimes starts the agent at the final state $\mathbf{s}_g$, and sometimes starts the agent in between. In fact, prior work has formally shown that the optimal initial state distribution is the marginal state distribution of the optimal policy [17]. Under somewhat strong assumptions,[3] this is precisely what our waypoint sampling achieves. We refer the reader for the discussion and experiments in Appendix. A.

4.2 A Practical Implementation

We now describe how we implement the policy updates (M-step) and waypoint sampling (E-step). For the policy updates, we simply apply C-learning [8]. The main challenge is waypoint sampling, which requires sampling a potentially high-dimensional waypoint distribution. While prior work [11] has approached such problems by fitting high-capacity generative models, our approach will avoid such generative models and instead use importance sampling.

Let $b(\cdot)$ be the background distribution, which we take to be the replay buffer. Our algorithm corresponds to the following two-step procedure. First, we sample a batch of waypoints from the buffer. Then, we sample one waypoint from within that batch using the normalized importance weights, $\frac{q(\mathbf{s}_w|\mathbf{s}_g,\mathbf{s}_0)}{b(\mathbf{s}_w)}$. We can estimate these importance weights using the classifier learned by C-learning (Eq. 3):

Lemma 3. *We can write the importance weights in terms of this value function*

$$\frac{q(\mathbf{s}_w \mid \mathbf{s}_g, \mathbf{s}_0)}{b(\mathbf{s}_w)} = \frac{C_\theta(\mathbf{s}_w, \mathbf{s}_g)}{1 - C_\theta(\mathbf{s}_w, \mathbf{s}_g)} \frac{C_\theta(\mathbf{s}_0, \mathbf{s}_w)}{1 - C_\theta(\mathbf{s}_0, \mathbf{s}_w)} Z(\mathbf{s}_0, \mathbf{s}_g), \tag{4}$$

where $Z(\mathbf{s}_0, \mathbf{s}_g)$ is a normalizing constant.

See Appendix A.3 for a full derivation. Since we will eventually normalize these importance weights (over $\mathbf{s}_w$), we do not need to estimate $Z(\mathbf{s}_0, \mathbf{s}_g)$. We call our complete algorithm *C-Planning*, and summarize it in Alg. 1. Note that our algorithm requires changing only a single line of pseudocode: during data collection, we command the agent to the intermediate waypoint $\mathbf{s}_w$ instead of the final goal $\mathbf{s}_g$. The sampling procedure in Alg. 2 is scalable and easy to implement.

The core idea of our method is to sample waypoints enroute to the goal. Of course, at some point we must command the agent to directly reach the goal. We introduce a hyperparameter n_g to indicate how many times we resample the intermediate waypoint before commanding the agent to reach the goal. This parameter is not the planning horizon, which is always set to 2.

[3]The assumptions are that *(1)* the policy always reaches the commanded waypoint, and that *(2)* that the goal-reaching probabilities $p_{\text{GEOM}}^{\pi(\cdot|\cdot,\cdot)}$ reflect the probability that the *optimal* policy reach some state.

Algorithm 1 **C-Planning** performs planning in data collection, modifies C-learning by $L5 \to L6-7$. The update for the policy and classifier ($L9$) is the same.

```
1: $\mathcal{D} \leftarrow \emptyset, N_g, \epsilon_d$
2: for $0 \le i \le N$ do
3:     Set $n_g \leftarrow 0$ if new episode
4:     $\boldsymbol{s}_0 \sim p_0(\boldsymbol{s}_0), \boldsymbol{s}_g \sim p_g(\boldsymbol{s}_g)$
5:     ~~$\tau \sim p^\pi(\tau \mid \boldsymbol{s}_0, \boldsymbol{s}_g)$~~
6:     $\boldsymbol{s}_w, n_g \leftarrow$ CPLANNING($n_g, C$)
7:     $\tau \leftarrow (\tau_1 \sim p^\pi(\tau \mid \boldsymbol{s}_0, \boldsymbol{s}_w),$
           $\tau_2 \sim p^\pi(\tau \mid \boldsymbol{s}_0, \boldsymbol{s}_w))$
8:     $\mathcal{D} \leftarrow \mathcal{D} \cup \{\tau\}$
9:     $\pi, C \leftarrow$ CLEARNING($\mathcal{D}, \pi, C$)
```

Algorithm 2 **C-Planning** samples the intermediate waypoints, then command the agent to reach them.

```
1: $n_g$: number of waypoints agent has reached in an episode
2: function CPLANNING($n_g, C$)
3:     if ($t = 0$ or $d(\boldsymbol{s}_t, \boldsymbol{s}_w) \le \epsilon_d$) and $n_g \le N_g$ then
4:         $n_g \leftarrow n_g + 1$
5:         Sample M waypoints $\boldsymbol{s}_w^{(i)} \sim$ Buffer($\boldsymbol{s}_w$)
6:         Compute distances for each candidate waypoint:
           $d^{(i)} \leftarrow \log \frac{C(F=1|\boldsymbol{s}_w, \boldsymbol{s}_g)}{C(F=0|\boldsymbol{s}_w, \boldsymbol{s}_g)} + \log \frac{C(F=1|\boldsymbol{s}_0, \boldsymbol{s}_w)}{C(F=0|\boldsymbol{s}_0, \boldsymbol{s}_w)})$
7:         $\boldsymbol{s}_w \propto$ SOFTMAX($d^{(i)}$)
8:     else
9:         Set $\boldsymbol{s}_w \leftarrow \boldsymbol{s}_g$
10: return $\boldsymbol{s}_w, n_g$
```

5 Experiments

Our experiments study whether C-Planning can compete with prior goal-conditioned RL methods both on benchmark tasks and on tasks designed to pose a significant planning and exploration challenge. Our first experiments use these tasks to compare C-Planning to prior methods for goal-conditioned RL. We then study the importance of a few design decisions through ablation experiments. The benefits of C-Planning, compared with prior methods, are especially pronounced on these challenging tasks. To provide more intuition for why C-Planning outperforms prior goal-conditioned RL methods, we plot the gradient norm for the actor and critic functions, finding that C-Planning enjoys larger norm gradients for the critic and smaller variance of the gradient for the actor. Ablation experiments study the number of waypoints used in training. We finally measure the inference time of choosing actions, finding that C-Planning is up to $4.7\times$ faster than prior methods that perform search at test time.

Environments. We investigate environments of various difficulty, some of which we visualize in in Fig. 3. The first set of environments is taken directly from the Metaworld benchmark suite [37], a common benchmark for goal-conditioned RL. These tasks are challenging because of the manipulation skills required to push and reorient objects. While prior goal-conditioned RL methods do solve some of these tasks (e.g., `Push` and `Reach`), they stand at the edge of the capabilities of current methods. The second set of environments, 2D navigation mazes, are designed to stress-test the planning capabilities of different methods. Successfully solving these tasks requires reasoning over long horizons of up to 50 steps. These are challenging because, unlike many benchmarks, they require non-greedy exploration: greedy strategies get stuck in local optima. Our final set of environments combine the challenges of these environments. We extend the tasks in Metaworld to involve *sequential* manipulation of *multiple* objects, similar to the tasks in prior work [34]. For example, the `Obstacle-Drawer-Close` task require the robotic arm to manipulate multiple objects in sequence, first pushing an object and then opening a drawer. Solving these tasks are difficult because the agent is given no demonstrations, no reward shaping, and no manually specified distance metrics.

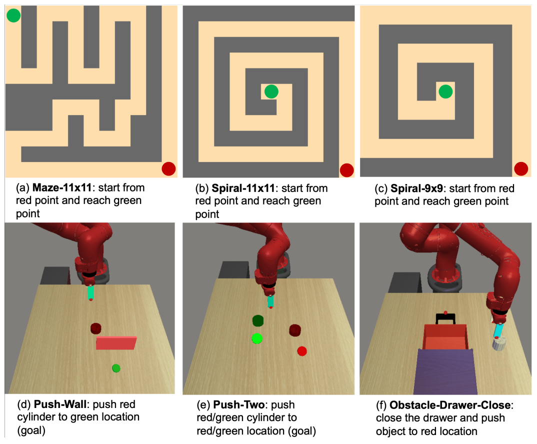
(a) **Maze-11x11**: start from red point and reach green point
(b) **Spiral-11x11**: start from red point and reach green point
(c) **Spiral-9x9**: start from red point and reach green point
(d) **Push-Wall**: push red cylinder to green location (goal)
(e) **Push-Two**: push red/green cylinder to red/green location (goal)
(f) **Obstacle-Drawer-Close**: close the drawer and push object to red location

Figure 3: **Environments**: Visualization of the 2D navigation maze environments *(top row)* and robotics manipulation tasks *(bottom row)*.

Baselines. We will compare C-Planning to a number of baselines. C-Learning is a recent goal-conditioned RL method that does not perform planning. C-Learning performs the same gradient updates as C-Planning, and only differs in how experience is collected. Comparing to C-Learning

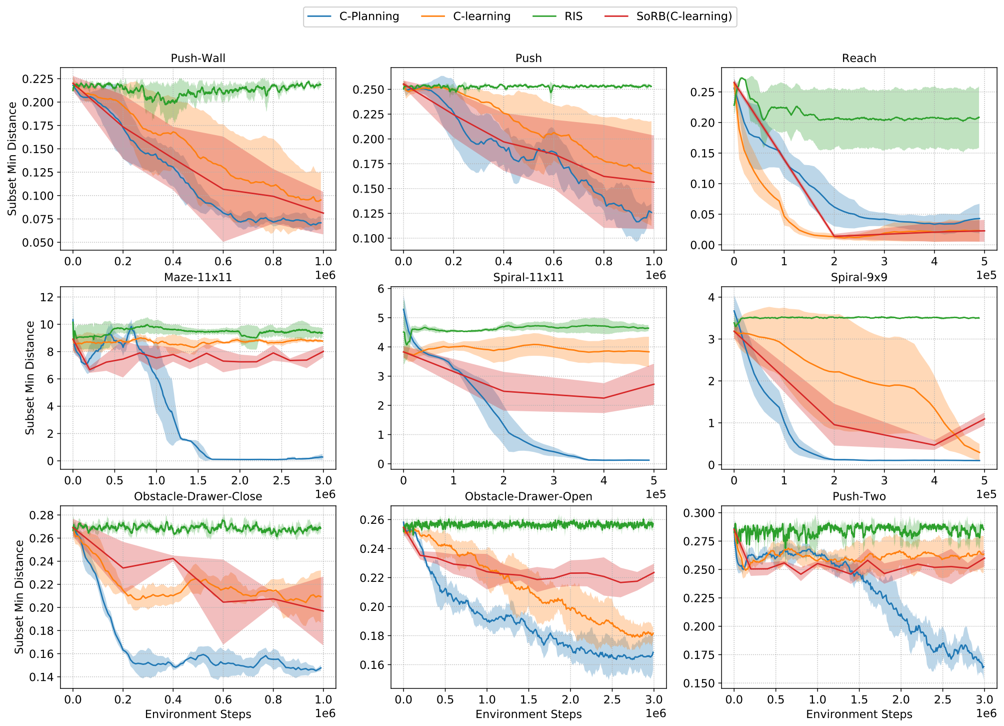

Figure 4: **Comparison of goal-conditioned RL methods**: We compare C-Planning to prior goal-conditioned RL algorithms on various tasks: *(top)* 2D maze navigation, *(middle)* benchmark manipulation tasks from Meta-world, and *(bottom)* robot manipulation tasks that require long horizon planning. For each task, we record the Euclidean distance to the goal, taking the minimum distance within an episode. All all but the easiest task (`Reach`), C-Planning outperforms all prior methods, including those that perform planning. Only C-Planning is able to solve the most challenging navigation and manipulation tasks.

will allow us to identify the marginal contribution of our curriculum of waypoints. The second baseline is SoRB [7], a goal-conditioned RL method that performs search at test-time, rather than training time. While SoRB was originally implemented using Q-learning, we find that a version based on C-learning worked substantially better, so we use this stronger baseline in our experiments. Comparing against SoRB will allow us to study the tradeoffs between performing search during training versus testing. Because SoRB performs search at testing, it is considerably more expensive to deploy in terms of computing. Thus, even matching the performance of SoRB, without incurring the computational costs of deployment would be a useful result. The third baseline is an ablation of our method designed to resemble RIS [3]. Like C-Planning, RIS performs search during training and not testing, but the search is used differently from C-Planning. Whereas C-Planning uses search to collect data, leaving the gradient updates unchanged, RIS modifies the RL objective to include an additional term that resembles behavior cloning. Our comparison with RIS thus allows us to study how our method for using the sampled waypoints compares to alternative methods to learn from those same sampled waypoints.

5.1 Comparison with Prior Goal-Conditioned RL Methods

To compare C-Planning to prior goal-conditioned RL algorithms, we start with three tasks from the MetaWorld suite: `Reach`, `Push`, `Push-Wall`. While C-Planning learns slightly slower than prior methods on the easiest task (`Reach`), we see a noticeable improvement over prior methods on the more challenging `Push` and `Push-Wall` tasks, tasks that require reasoning over longer horizons to solve. This observation suggests that that the waypoint sampling performed by C-Planning might be especially beneficial for solving tasks that require planning over long horizons.

To test the planning capabilities of C-Planning, we design a series of 2D navigation mazes. While the underlying locomotion motions are simple, these tasks post challenges for planning and are designed so that a greedy planning algorithm would fail. On all three mazes, C-Planning learns faster than all baselines and achieves a lower asymptotic distance to the goal, as compared to the baselines. On the most challenging maze, `Maze-11x11`, only our method is able to make any learning progress.

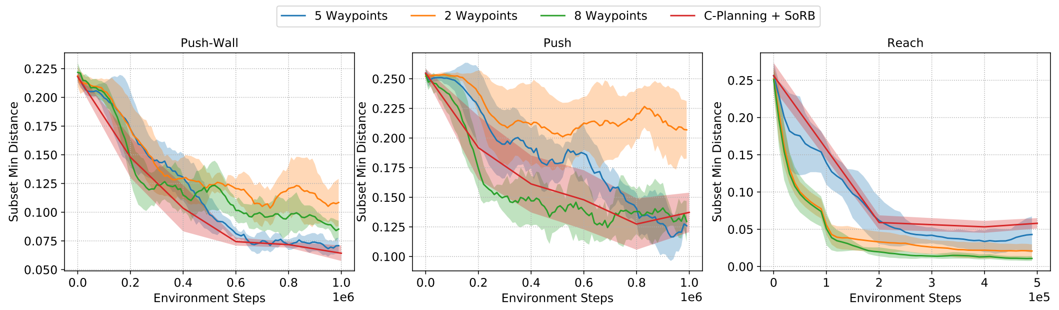

Figure 5: **Planning at Training Versus Testing:** We ablate the number of intermediate waypoints used to reach the goal, and compare against a variant of our method that performs planning during both training and testing ("C-Planning + SoRB"). This variant does not improve performance, indicating that the feedforward policy learned by C-Planning has already "internalized" these planning capabilities.

Of particular note is the comparison with SoRB, which uses the same underlying goal-conditioned RL algorithm as C-Planning, but differs in that it performs search at testing, rather than training. While SoRB performs better than C-learning, which does not perform planning, SoRB consistently performs worse than C-Planning. This observation suggests that C-Planning is not just amortizing the cost of performing search. Rather, these results suggest that incorporating search into training can produce significantly larger gains than incorporating search into the deployed policy.

As the last set of experiments, study higher dimensional tasks that require long-range planning. We design three environments: `Obstacle-Drawer-Close`, `Obstacle-Drawer-Open` and `Push-Two`. These tasks have a fairly large dimension (15 to 18) and require sequential manipulation of multiple objects. C-Planning outperforms all baselines on all three tasks. While C-learning makes some learning progress, SoRB (which modifies C-learning with search at test time) does not improve the results. RIS, which also samples waypoints during training but uses those waypoints differently, does not make any learning progress on these tasks. Taken together, these results suggest that waypoint sampling (as performed by C-Planning) can significantly aid in the solving of complex manipulation tasks, but only if that waypoint sampling is performed during training. Moreover, the comparison with RIS suggests that those waypoints should be used to collect new data, rather than to augment the policy learning objective. To the best of our knowledge, C-Planning is the first method to learn manipulation behavior of this complexity without additional assumptions (such as dense rewards or demonstrations).

5.2 Why Does C-Planning Work?

We ran many additional experiments to understand *why* C-Planning works so well. To start, we run ablation experiments to answer two questions: *(1)* Although C-Planning only applies planning at training time, does *additionally* performing planning at test time further improve performance? *(2)* How many times should we resample the waypoint before directing the agent to the goal?

Fig 5 shows the results of ablation experiments. C-Planning performs the same at test-time with and without SoRB-based planning, suggesting that our training-time planning procedure already makes the policy sufficiently capable of reaching distant goals, such that it does not benefit from additional test-time planning. Fig. 5 also shows that C-Planning is relatively robust to n_g, the number of waypoints used before directing the agent to the goal. For all choices of this hyperparameter, C-Planning still manages to solve all the tasks.

Gradient analysis. We provide an empirical observation why C-Planning is better than vanilla goal-conditioned RL methods. We hypothesize that, by introducing waypoints, C-Planning provides a better learning signal, increasing the gradient norm for training the critic and decreasing the variance of the gradient for the policy. While prior work has theoretically analyzed the importance of gradient norms for RL [1], we will provide some empirical evidence for the importance of controlling gradient norms. In Fig. 7, we plot the norm of the gradient (mean and variance) of both C-Planning and C-Learning using the Maze-11x11 environment, finding that our method increases the norm of the critic norm and decreases the variance of the actor gradient norm.

Test-time latency. One important advantage of C-Planning is that it enables direct execution at test time, does not require any online planning. At test time, we directly set the agent's goal to the

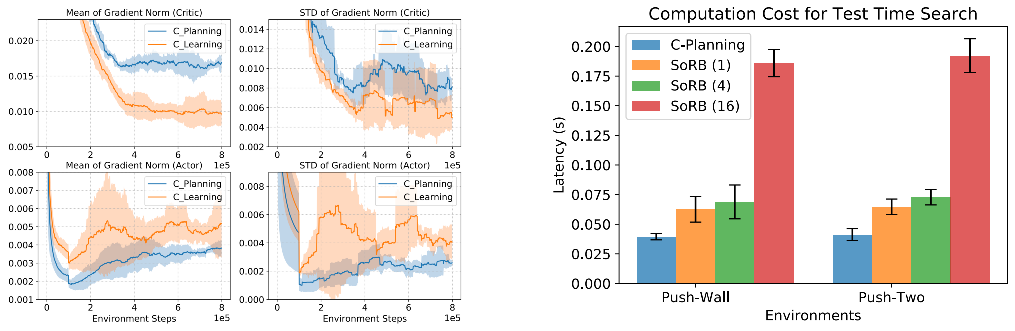

Figure 7: **Gradient Analysis and Computation Cost:** *(Left)* Mean and standard deviation of the gradient norm for the actor and critic networks. C-Planning has a larger critic gradient norm than C-Learning. C-Planning shows a smaller variance comparing to C-Learning. *(Right)* Computation cost in latency of C-Planning and SoRB with various number of waypoints.

final goal state without commanding any intermediate states. This makes C-Planning considerably different from previous planning methods, such as SoRB [7]. Not only does our method achieve a higher return, but it does so with significantly less computing at deployment. We measure the execution time at evaluation time for C-Planning and SoRB with a various number of waypoints in Fig. 7. C-Planning is 1.74$\times$ faster than planning over four waypoints and 4.71$\times$ faster than planning over 16 waypoints in `Push-Wall` environment.

Visualizing the training dynamics. To further gain intuition into the mechanics of our method, we visualize how the distribution over waypoints changes during training of the 2D navigation of the four rooms environment. Fig. 6 shows the sampled waypoints. The value functions (i.e., future state classifiers) are randomly initialized at the start of training, so the waypoints sampled are roughly uniform over the state space. As training progresses, the distribution over waypoints converges to the states that an optimal policy would visit enroute to the goal. While we have only shown one goal here, our method trains the policy for reaching all goals. This set of experiments provides intuition of how C-Planning works as the distribution of waypoints shrinks from a uniform distribution to the single path connecting start state and goal state. This visualization is aligned with our theory (Eq. 2), which says that the distribution of waypoints should resemble the states visited by the optimal policy.

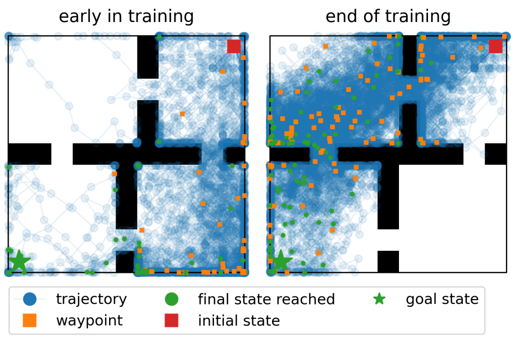

Figure 6: **Waypoint sampling:** *(Left)* Early in training, the agent samples waypoints closer to the initial state. *(Right)* At convergence, waypoints are evenly distributed along states visited by the optimal policy, as predicted by our theory.

6 Conclusion

In this paper, we introduced C-Planning, a method for incorporating search into the training of goal-conditioned agents. The method enables the automatic generation of a curriculum over intermediate waypoints. Unlike prior methods, our approach avoids the computational complexity of performing search at test time. C-Planning is simple to implement on top of existing goal-conditioned RL algorithms and achieves state-of-the-art results on a range of navigation and manipulation tasks.

Our work suggests a few directions for future work. First, C-Planning samples waypoint that correspond to the optimal state distribution, which is proovably optimal in some settings [17]. How might similar ideas be applied to reward-driven tasks? Second, while this paper studied sought to learn agents for reaching distant goals, we assumed that those distant goals were provided as part of the task definition. Goal-conditioned algorithms that can propose their own distant goals, while already studied in some prior work [11, 28], remains an important direction for future work.

ACKNOWLEDGEMENTS

This material is supported by the Fannie and John Hertz Foundation, NSF GRFP (DGE1745016), NSF CISE Expeditions Award CCF-1730628, and gifts from Alibaba, Amazon Web Services, Ant Financial, CapitalOne, Ericsson, Facebook, Futurewei, Google, Intel, Microsoft, Nvidia, Scotiabank, Splunk and VMware.

REFERENCES

[1] Alekh Agarwal, Sham M Kakade, Jason D Lee, and Gaurav Mahajan. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *Journal of Machine Learning Research*, 22 (98):1–76, 2021.

[2] Hagai Attias. Planning by probabilistic inference. In *International Workshop on Artificial Intelligence and Statistics*, pp. 9–16. PMLR, 2003.

[3] Elliot Chane-Sane, Cordelia Schmid, and Ivan Laptev. Goal-conditioned reinforcement learning with imagined subgoals. In *International Conference on Machine Learning*, pp. 1430–1440. PMLR, 2021.

[4] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[5] Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. *Advances in Neural Information Processing Systems*, 32:15324–15335, 2019.

[6] Benjamin Eysenbach, Shixiang Gu, Julian Ibarz, and Sergey Levine. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. In *International Conference on Learning Representations*, 2018.

[7] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[8] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. C-learning: Learning to achieve goals via recursive classification. In *International Conference on Learning Representations*, 2020.

[9] William Fedus, Carles Gelada, Yoshua Bengio, Marc G Bellemare, and Hugo Larochelle. Hyperbolic discounting and learning over multiple horizons. *arXiv preprint arXiv:1902.06865*, 2019.

[10] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *Conference on robot learning*, pp. 482–495. PMLR, 2017.

[11] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pp. 1515–1528. PMLR, 2018.

[12] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Conference on Robot Learning*, pp. 1025–1037. PMLR, 2020.

[13] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29:4565–4573, 2016.

[14] David Hsu, Jean-Claude Latombe, and Hanna Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *The International Journal of Robotics Research*, 25(7):627–643, 2006.

[15] Zhiao Huang, Fangchen Liu, and Hao Su. Mapping state space using landmarks for universal goal reaching. *arXiv preprint arXiv:1908.05451*, 2019.

[16] Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pp. 1094–1099. Citeseer, 1993.

[17] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*. Citeseer, 2002.

[18] Aviral Kumar, Abhishek Gupta, and Sergey Levine. Discor: Corrective feedback in reinforcement learning via distribution correction. *Advances in Neural Information Processing Systems*, 33, 2020.

[19] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

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

[21] Xingyu Lin, Harjatin Singh Baweja, and David Held. Reinforcement learning without ground-truth state. *arXiv preprint arXiv:1905.07866*, 2019.

[22] Aleksandr Mikhailovich Lyapunov. The general problem of the stability of motion. *International journal of control*, 55(3):531–534, 1992.

[23] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on Robot Learning*, pp. 1113–1132. PMLR, 2020.

[24] Xiangyun Meng, Nathan Ratliff, Yu Xiang, and Dieter Fox. Scaling local control to large-scale topological navigation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 672–678. IEEE, 2020.

[25] Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. *Advances in Neural Information Processing Systems*, 32: 2318–2328, 2019.

[26] Soroush Nasiriany, Vitchyr Pong, Steven Lin, and Sergey Levine. Planning with goal-conditioned policies. *Advances in Neural Information Processing Systems*, 32:14843–14854, 2019.

[27] Silviu Pitis, Harris Chan, Stephen Zhao, Bradly Stadie, and Jimmy Ba. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In *International Conference on Machine Learning*, pp. 7750–7761. PMLR, 2020.

[28] Vitchyr Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. In *International Conference on Machine Learning*, pp. 7783–7792. PMLR, 2020.

[29] Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *Twenty-third international joint conference on artificial intelligence*, 2013.

[30] Zhizhou Ren, Kefan Dong, Yuan Zhou, Qiang Liu, and Jian Peng. Exploration via hindsight goal generation. *Advances in Neural Information Processing Systems*, 32:13485–13496, 2019.

[31] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In *International Conference on Learning Representations*, 2018.

[32] Yannick Schroecker and Charles Isbell. Universal value density estimation for imitation learning and goal-conditioned reinforcement learning. *arXiv preprint arXiv:2002.06473*, 2020.

[33] Dhruv Shah, Benjamin Eysenbach, Gregory Kahn, Nicholas Rhinehart, and Sergey Levine. Ving: Learning open-world navigation with visual goals. *arXiv preprint arXiv:2012.09812*, 2020.

[34] Avi Singh, Albert Yu, Jonathan Yang, Jesse Zhang, Aviral Kumar, and Sergey Levine. Cog: Connecting new skills to past experience with offline reinforcement learning. *arXiv preprint arXiv:2010.14500*, 2020.

[35] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. A generalized path integral control approach to reinforcement learning. *The Journal of Machine Learning Research*, 11:3137–3181, 2010.

[36] Emanuel Todorov. Linearly-solvable markov decision problems. In *Advances in neural information processing systems*, pp. 1369–1376, 2007.

[37] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pp. 1094–1100. PMLR, 2020.

[38] Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E Gonzalez, and Yuandong Tian. Bebold: Exploration beyond the boundary of explored regions. *arXiv preprint arXiv:2012.08621*, 2020.

[39] Tianjun Zhang, Paria Rashidinejad, Jiantao Jiao, Yuandong Tian, Joseph Gonzalez, and Stuart Russell. Made: Exploration via maximizing deviation from explored regions. *arXiv preprint arXiv:2106.10268*, 2021.

[40] Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.

A Proofs and Additional Analysis

A.1 Deriving the Lower Bound (Proof of Lemma 1)

We first formulate the sampling procedure on starting states s_0, waypoints s_w, goals s_g and the corresponding time horizon variable t_1 and t_2. Then we derive a lower bound on the target log density of Eq. 2. We then show that optimizing the lower bound through an EM procedure is equivalent to breaking the goal-reaching task into a sequence of easier sub-problems. Finally, we wrapped up this section with a practical algorithm.

Data Generation Process. The generative model for which inference corresponds to our planning procedure can be formulated as follows. The episode starts by sampling an initial state $s_0 \sim p_0(s_0)$. Then it samples a geometric random variable $t_1 \sim Geom(1-\gamma)$ and roll out the policy $\pi(a \mid s, s_g)$ for exactly t_1 steps, starting from state s_0. We define s_w to be the state where we end up (i.e., $s_w \triangleq s_{t_1}$). Thus, s_w is sampled $s_w \sim p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_{t+} \mid s_0)$. We then sample another geometric random variable $t_2 \sim Geom(1-\gamma)$ and roll out the policy $\pi(a \mid s, s_g)$ for exactly t_1 steps, starting from state s_w. We define s_g to be the state where we end up (i.e., $s_g \triangleq s_{t_1+t_2}$). Thus, s_g is sampled $s_g \sim p_{Geom}^{\pi(\cdot|\cdot,s_g)}(s_{t+} \mid s_w)$. Note that the time index of the final state s_g is a sample from a negative binomial distribution: $t_1 + t_2 \overset{d}{=} NB(p = 1-\gamma, n = 2)$. We can equivalently express the sampling of s_g as $s_g \sim p_{NegBinom}^{\pi(\cdot|\cdot,s_g)}(s_{t+} \mid s_0)$.

Inference process. Under the formulation of the data generation process above, we then aim to answer the following question in the inference procedure: what intermediate states would a policy visit if it eventually reached the goal state s_g? Formally, we will estimate a distribution $q(s_w \mid s_0, s_g) \approx p(s_w \mid s_0, s_g)$.

We learn $q(s_w \mid s_0, s_g)$ by optimizing a evidence lower bound on our main objective (Eq. 2).

$$\log p_{\text{NEGBINOM}}^{\pi(\cdot|\cdot,s_g)}(s_{t+} = s_g \mid s_0) \tag{5}$$

$$= \log \int p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_{t+} = s_g \mid s_w) p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_w \mid s_0) ds_w \tag{6}$$

$$= \log \int p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_{t+} = s_g \mid s_w) p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_w \mid s_0) \frac{q(s_w \mid s_g, s_0)}{q(s_w \mid s_g, s_0)} ds_w \tag{7}$$

$$\geq \int q(s_w \mid s_g, s_0) \Big(\log p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_{t+} = s_g \mid s_w) + \tag{8}$$

$$\log p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_w \mid s_0) - \log q(s_w \mid s_g, s_0) \Big) ds_w \tag{9}$$

$$\triangleq \mathcal{L}(\pi, q(s_w \mid s_g, s_0)). \tag{10}$$

Note that s_g is conditionally independent of s_0 given s_w, so the $p^\pi(s_{t+} = s_g \mid s_w)$ terms on the RHS need not be conditioned on s_0. The evidence lower bound, $\mathcal{L}$, depends on two quantities: the goal-conditioned policy and the distribution over waypoints. The objective for the goal-conditioned policy is to maximize the probabilities of reaching the waypoint and reaching the final state. The objective for the waypoint distribution is to select waypoints s_w that satisfy two important properties: the current policy should have a high probability of successfully navigating from the initial state to the waypoint and from the waypoint to the final goal. Note that the optimal choice for the waypoint distribution automatically depends on the current capabilities of the goal-conditioned policy.

Before optimizing the lower bound, we introduce a subtle modification to the lower bound:

$$\mathcal{L}_2(\pi, q(s_w \mid s_g, s_0)) \triangleq \int q(s_w \mid s_g, s_0) \Big(\log p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_{t+} = s_g \mid s_w) + \tag{11}$$

$$\log p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_w)}(s_w \mid s_0) - \log q(s_w \mid s_g, s_0) \Big) ds_w. \tag{12}$$

The difference, highlighted in orange, is that the probability of reaching the waypoint is computed for a goal-conditioned policy that is commanded to reach that waypoint, rather than the final goal. We show that this new objective is also an evidence lower bound on the same goal-reaching objective

(Eq. 2), but modified such that the sequence of *commanded* goals is treated as an additional latent variable.

A.2 THE OPTIMAL WAYPOINT DISTRIBUTION (PROOF OF LEMMA 2

This section proves Lemma 2.

Proof. Recall that our goal is to solve the following maximization problem:

$$\max_{q(s_w|s_g,s_0)} \mathbb{E}_{q(s_w|s_g,s_0)}\left[\log p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_{t+}=s_g \mid s_w) + \log p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_w \mid s_0) - \log q(s_w \mid s_g, s_0)\right]. \tag{13}$$

Note that the waypoint distribution must integrate to one. The Lagrangian can be written as

$$\mathbb{E}_{q(s_w|s_g,s_0)}\left[\log p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_{t+}=s_g \mid s_w) + \log p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_w \mid s_0) - \log q(s_w \mid s_g, s_0)\right] + \tag{14}$$

$$\lambda\left(\int q(s_w \mid s_0, s_g) ds_w - 1\right), \tag{15}$$

where λ is a Lagrange multiplier. We then take the derivative with respect to $q(s_w \mid s_g, s_0)$:

$$\frac{d}{dq(s_w \mid s_0, s_g)} = \frac{-q(s_w \mid s_0, s_g)}{q(s_w \mid s_0, s_g)} + \log p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_{t+}=s_g \mid s_w) + \tag{16}$$

$$\log p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_w \mid s_0) - \log q(s_w \mid s_g, s_0) + \lambda \tag{17}$$

$$= -1 + \log p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_{t+}=s_g \mid s_w) + \log p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_w \mid s_0) - \tag{18}$$

$$\log q(s_w \mid s_g, s_0) + \lambda. \tag{19}$$

We then set this derivative equal to zero and solve for $q(s_w \mid s_g, s_0)$:

$$q(s_w \mid s_g, s_0) = e^{\lambda-1} p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_{t+}=s_g \mid s_w) p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_w \mid s_0).$$

Finally, we determine the value of λ such that $q(s_w \mid s_0, s_g)$ integrates to one. We can then express the optimal waypoint distribution as follows:

$$q^*(s_w \mid s_g, s_0) = \frac{p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_g \mid s_w) p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_w)}(s_w \mid s_0)}{\int p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_g \mid s_w') p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_w)}(s_w' \mid s_0) ds_w'}.$$

□

A.3 ESTIMATING IMPORTANCE WEIGHTS (PROOF OF LEMMA 3)

This section proves Lemma 3.

Proof. Define the normalizing constant as follows

$$Z(s_0, s_g) = \frac{b(s_g)}{\int p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_g \mid s_w') p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_w)}(s_w' \mid s_0) ds_w'}.$$

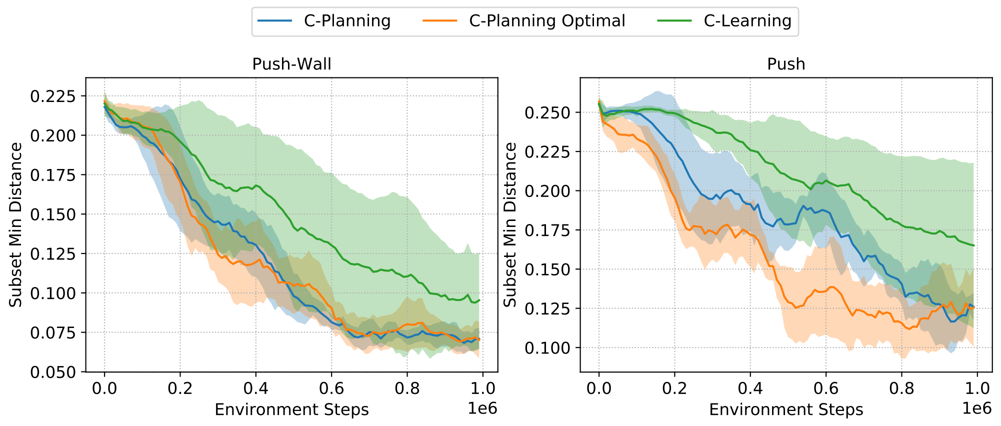

Figure 8: **Oracle experiment with the optimal generative model**: Goal-conditioned RL learns the task much faster if we can sample the waypoints from the expert policy's marginal state distribution (C-Planning Optimal). Compared to planning with optimal state density distribution, planning using a learned classifier (C-Planning) shows comparable performance.

Substituting $Z(s_0, s_g)$ into the RHS of Eq. 4 and simplifying the result, we show that it equals the LHS of Eq. 4:

$$\frac{C_\theta(s_w, s_g)}{1 - C_\theta(s_w, s_g)} \frac{C_\theta(s_0, s_w)}{1 - C_\theta(s_0, s_w)} Z(s_0, s_g) \tag{20}$$

$$= \frac{C_\theta(s_w, s_g)}{1 - C_\theta(s_w, s_g)} \frac{C_\theta(s_0, s_w)}{1 - C_\theta(s_0, s_w)} \frac{b(s_g)}{\int p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_g \mid s_w') p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_w)}(s_w' \mid s_0) ds_w'} \tag{21}$$

$$= \frac{p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_{t+} = s_g \mid s_w)}{\cancel{b(s_g)}} \frac{p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_w)}(s_{t+} = s_w \mid s_0)}{b(s_w)} \frac{\cancel{b(s_g)}}{\int p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_g \mid s_w') p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_w)}(s_w' \mid s_0) ds_w'} \tag{22}$$

$$= \frac{p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_{t+} = s_g \mid s_w) p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_w)}(s_{t+} = s_w \mid s_0)}{\int p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_g)}(s_g \mid s_w') p_{\text{GEOM}}^{\pi(\cdot|\cdot,s_w)}(s_w' \mid s_0) ds_w'} \frac{1}{b(s_w)} \tag{23}$$

$$= \frac{q(s_w \mid s_0, s_g)}{b(s_w)}. \tag{24}$$

□

B Additional Experiments

B.1 An Oracle Experiment

We run an experiment to confirm the well-known result [17] that the optimal initial state distribution for learning a task is the state distribution of that task's optimal policy. Intuitively, if we could sample exactly from the state distribution of the optimal policy, then an RL agent could learn to reach goals more quickly. This is not surprising as it resembles behavioral cloning of the optimal policy. In this oracle experiment, it is assumed we are given the access to an optimal policy. We achieve this by following the sampling procedure from Algorithm. 1, but replace the classifier C with that learned from the optimal policy C^*. We then perform RL to reach waypoints and then the final goal. Results in Fig. 8 show that using the optimal policy as the initial state distribution results in faster learning than using the original state distribution. We also visualize the state density distribution of the optimal policy to provide more qualitative intuition on how the algorithm works.

In addition, we are also interested in measuring how well does planning through a learned model (C-Planning) performs if we don't have access to the optimal generative model. Results in Fig. 8 also show that C-Planning achieves performance comparable to planning via optimal generative model in the Push and the Push-Wall environment.

B.2 Visualization of State Density Map of Optimal policy

We conduct this experiment on a 2D navigation task shown in Fig. 8 (left), where we have also visualized the original initial state distribution, the state distribution of an optimal policy, and the goal state. To conduct this experiment, we apply a state-of-the-art goal-conditioned RL algorithm (C-learning) in the two settings with different initial state distributions. For fair evaluation, we evaluate the policies learned in both settings using the original initial state distribution. The results shown in Fig. 8 (right) show that starting from the optimal initial state distribution results in 2.4x faster learning.

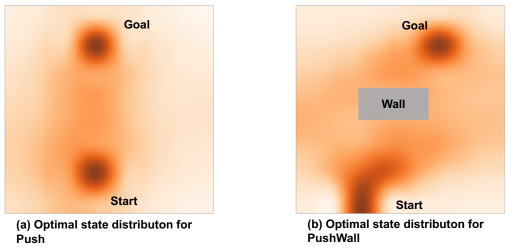

Figure 9: *(Left)*An agent must navigate from the `start` state to the `goal` state. The heatmap visualizes the marginal state distribution of the optimal policy.

C Experimental Details

In this section, We provide the essential hyperparameters for reproducing our experiments in this section. We also introduce the hyperparameters used in baselines and provide a detailed description of environmental design.

C.1 Implementation Details

We introduce the hyperparameters used in C-Planning. Note that in C-Learning, only a classifier on $(\boldsymbol{s}_t, \boldsymbol{s}_{t+}, \boldsymbol{a}_t)$ is needed. In C-Planning, in order to sample waypoints from the distribution, an additional classifier on $(\boldsymbol{s}_t, \boldsymbol{s}_{t+})$ is needed. We also introduce two hyperparameters: Maximum Steps Reaching Goal (N_g in Alg.1) forces the agent to change the intermediate goal if the original goal hasn't been reached for some steps; Distance Threshold Reaching Goal (ϵ_d in Alg.1) for determing whether a goal is reached or not. The rest are the standard hyperparameters for SAC algorithm and we list here for reference.

Table 1: Hyperparameters used for C-Planning in all the environments in MetaWorld.

	Hyperparameter Value
Actor lr	0.0003
Action-State Critic lr	0.0003
State Critic lr	0.00003
Actor Network Size	(256, 256, 256)
Critic Network Size	(256, 256, 256)
Maximum Steps Reaching Goal	20
Distance Threshold Reaching Goal	0.05
Actor Loss Weight	1.0
Critic Loss Weight	0.5
Discount	0.99
Target Update Tau	0.005
Target Update Period	1
Number Waypoints	5
Goal Relabel Next	0.3
Goal Relabel Future	0.2

Note that we use a slightly different hyperparameters for 2D maze environment and we list below, all the other hyperparameters remains the same. Note that we follow the goal relabeling changes by C-Learning [8].

Table 2: Hyperparameters used for C-Planning in all the environments in 2D navigation maze.

	Hyperparameter Value
Distance Threshold Reaching Goal	1.0
Number Waypoints	8
Goal Relabel Next	0.5
Goal Relabel Future	0.0

C.2 Environments

We follow the envioronment design of [8] with only one noticeable difference: in the original environments of C-Learning, for the ease of training, the author set the initial position of objects to be relatively near the arm so the arm can easily push the object, getting a better learning signal. We intentionally set the initial state of object to be far away from the arm. This significantly increase the diffuculty of learning. We'll release these environment with the code.

C.3 Baselines

We also provide the hyperparameters associated with the baselines. The two baselines we care about: C-Learning and RIS. We summarize their hyperparameters in the table:

Table 3: Hyperparameters used for C-Learning in all the environments in MetaWorld.

	Hyperparameter Value
Actor lr	0.0003
Action-State Critic lr	0.0003
Actor Network Size	(256, 256, 256)
Critic Network Size	(256, 256, 256)
Actor Loss Weight	1.0
Critic Loss Weight	0.5
Discount	0.99
Target Update Tau	0.005
Target Update Period	1
Goal Relabel Next	0.3
Goal Relabel Future	0.2

Table 4: Hyperparameters used for RIS in all the environments in MetaWorld.

	Hyperparameter Value
epsilon	0.0001
Replay Buffer Goals	0.5
Distance Threshold	0.05
Alpha	0.1
Lambda	0.1
H lr	0.0001
Q lr	0.001
Pi lr	0.0001
Encoder lr	0.0001

D Visualization of the Learned Policy

In order to more intuitively visualize the behavior of the learned policy and emphasize the importance of the difficulty of the learning task, we plot the visualization of our learned policy for several snap shots in an episode for environment of `Push-Two` and `Obstacle-Drawer-Open`. We see that our method successfully guide the agent to learn the behavior of push the green object first, the push the red object; And first open the drawer then push the object to the desired location. We would

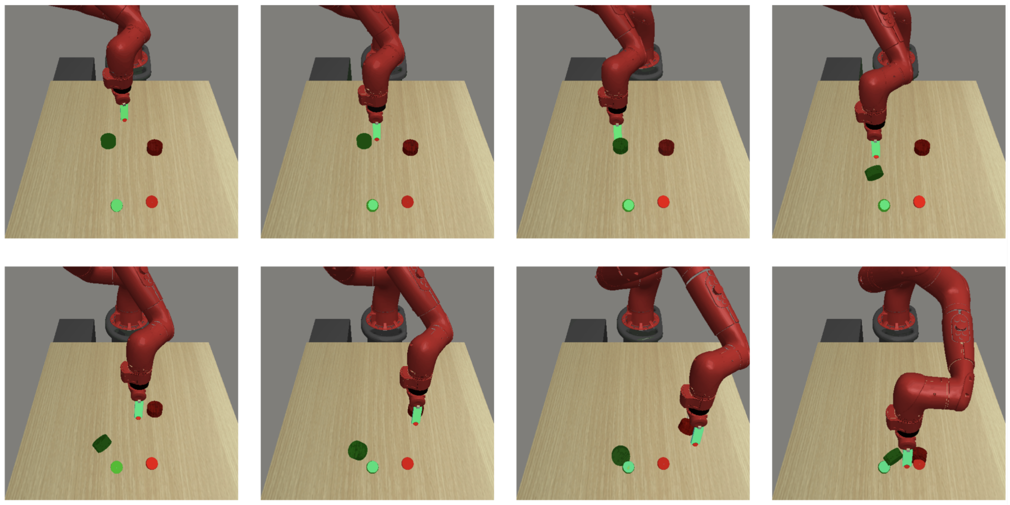

Figure 10: Visualization of the trained policy for the `Push-Two` environment. Our method successfully trains the agent to manipulate the two object in the sequential manner without any offline data, expert demonstration and reward shaping. Prior baselines all fail to demonstrate such behavior.

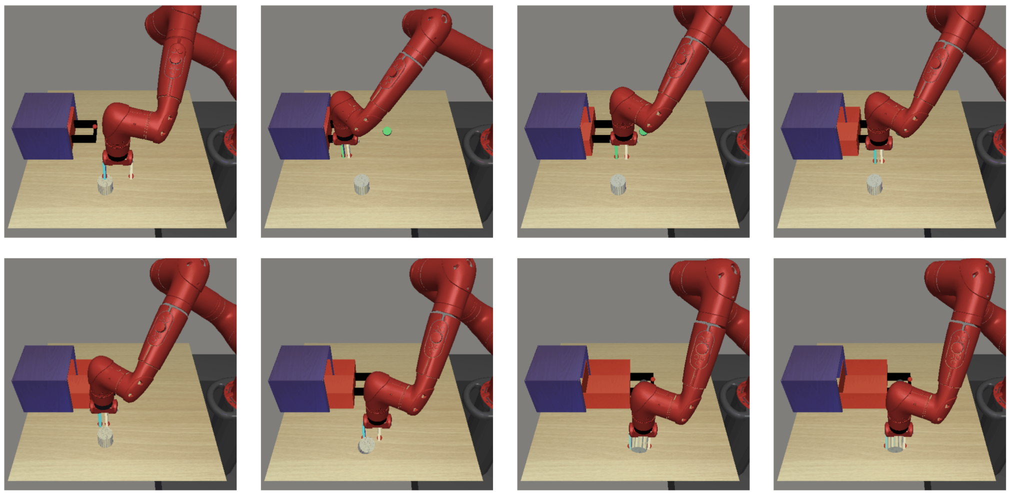

Figure 11: Visualization of the trained policy for the `Obstacle-Drawer-Open` environment. Our method successfully trains the agent to first close the drawer and then push the object to the desired location. We emphasize that such behavior is hard to obtain and most of the prior methods only manage to close the drawer.

like to emphasize that prior baselines all fail to demonstrate such behavior without any offline data, expert demonstration and reward shaping. The successfully learning of such behavior enables the robot to do more complex tasks without any human intervention.