EAT-C: Environment-Adversarial sub-Task Curriculum for RL

Shuang Ao [1] **Tianyi Zhou** [2 3] **Jing Jiang** [1] **Guodong Long** [1] **Xuan Song** [4] **Chengqi Zhang** [1]

Abstract

Reinforcement learning (RL) is inefficient on long-horizon tasks due to sparse rewards and its policy can be fragile to slightly perturbed environments. We address these challenges via a curriculum of tasks with coupled environments, generated by two policies trained jointly with RL: (1) a co-operative planning policy recursively decomposing a hard task into a coarse-to-fine sub-task tree; and (2) an adversarial policy modifying the environment in each sub-task. They are complementary to acquire more informative feedback for RL: (1) provides dense reward of easier sub-tasks while (2) modifies sub-tasks' environments to be more challenging and diverse. Conversely, they are trained by RL's dense feedback on sub-tasks so their generated curriculum keeps adaptive to RL's progress. The sub-task tree enables an easy-to-hard curriculum for every policy: its top-down construction gradually increases sub-tasks the planner needs to generate, while the adversarial training between the environment and RL follows a bottom-up traversal that starts from a dense sequence of easier sub-tasks allowing more frequent environment changes. We compare EAT-C with RL/planning targeting similar problems and methods with environment generators or adversarial agents. Extensive experiments on diverse tasks demonstrate the advantages of our method on improving RL's efficiency and generalization.

1. Introduction

Although RL achieves breakthrough success on some challenging tasks (Lillicrap et al., 2016; Mnih et al., 2015; Florensa et al.), it is highly inefficient when targeting long-horizon tasks due to the sparse rewards. Moreover, a policy trained in a specified environment can be sensitive to small changes in the deployed environment and thus generalizes poorly in practice. Hence, selecting or generating more informative tasks and environments to train an agent is essential to more efficient, robust, and versatile RL. In this paper, we mainly study goal-conditioned RL (Kaelbling, 1993) whose policy is trained to adapt to any given goal/task.

[1]University of Technology Sydney [2]University of Washington, Seattle [3]University of Maryland, College Park [4]Southern University of Science and Technology. Correspondence to: Shuang Ao <shuang.ao@students.tus.edu.au>, Tianyi Zhou <tianyizh@uw.edu>, Jing Jiang, Guodong Long, Chengqi Zhang <{jing.jiang, guodong.long, chengqi.zhang}@uts.edu.au>, Xuan Song <songx@sustech.edu.cn>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

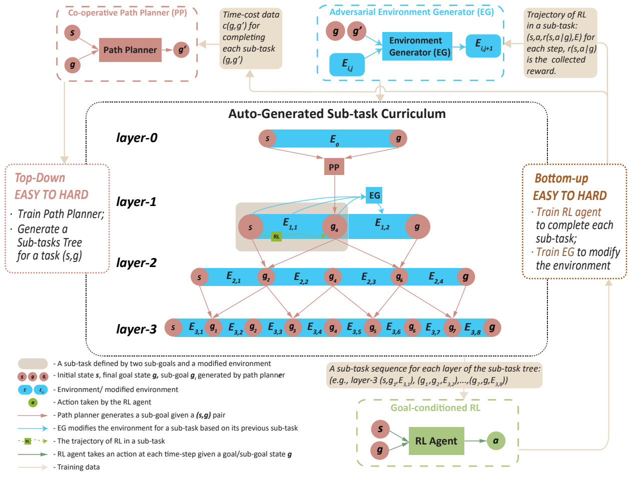

Co-operative Path Planner (PP)
s
g
Path Planner
g'
Time-cost data c(g,g') for completing each sub-task (g,g')
Adversarial Environment Generator (EG)
g
g'
$E_{i,j}$
Environment Generator (EG)
$E_{i,j+1}$
Trajectory of RL in a sub-task: (s,a,r(s,a|g),E) for each step, r(s,a|g) is the collected reward.
Auto-Generated Sub-task Curriculum
layer-0
layer-1
layer-2
layer-3
E_0
PP
EG
$E_{1,1}$
g_4
$E_{1,2}$
RL
$E_{2,1}$ g_2 $E_{2,2}$ g_4 $E_{2,3}$ g_6 $E_{2,4}$
$E_{3,1}$ g_1 $E_{3,2}$ g_2 $E_{3,3}$ g_3 $E_{3,4}$ g_4 $E_{3,5}$ g_5 $E_{3,6}$ g_6 $E_{3,7}$ g_7 $E_{3,8}$
Top-Down EASY TO HARD
· Train Path Planner;
· Generate a Sub-tasks Tree for a task (s,g)
Bottom-up EASY TO HARD
· Train RL agent to complete each sub-task;
· Train EG to modify the environment
- A sub-task defined by two sub-goals and a modified environment
- Initial state s, final goal state g, sub-goal g_i generated by path planner
- Environment/ modified environment
- Action taken by the RL agent
- Path planner generates a sub-goal given a (s,g) pair
- EG modifies the environment for a sub-task based on its previous sub-task
- The trajectory of RL in a sub-task
- RL agent takes an action at each time-step given a goal/sub-goal state g
- Training data
A sub-task sequence for each layer of the sub-task tree: (e.g., layer-3 $(s,g_1,E_{3,1}), (g_1,g_2,E_{3,2}),\dots,(g_7,g,E_{3,8})$)
Goal-conditioned RL
RL Agent
a

Figure 1: **Main structure of EAT-C:** The path-planner recursively generates a sub-task tree for a task (g_0, g), while the environment generator (EG) adversarially modifies the environment of each sub-task. RL agent is trained on a bottom-up curriculum and its collected data are used to train the path-planner and EG. (Larger version in Fig. 6)

The sparse reward problem has motivated reward shaping/relabeling and curiosity-driven exploration methods that provide dense reward. Hierarchical RL/planning (Elbanhawi & Simic, 2014; Nasiriany et al., 2019; Jurgenson et al., 2020; Ao et al., 2021; Pertsch et al., 2020) decomposes a complicated task/motion as a root-to-leaf path of sub-tasks on a search tree, where a higher-level task invokes lower-level ones. However, building the tree requires expensive hierarchical-partition of the whole state-action space and pre-defined sub-tasks, which can be either too easy or too hard for different stages of RL. Reward shaping (Laud, 2004) relies on heuristics or external guidance to provide dense rewards for exploring uncertain actions, rarely-visited states, or intermediate goals. Hindsight experience replay (Andrychowicz et al., 2017; Fang et al., 2019) relabels achieved states as pseudo-goals with nonzero rewards but they are usually redundant to provide informative feedback. Hence, how to automatically modify the sparse reward and generate sub-tasks to provide informative feedback is still an open problem and resolving it can lead to more efficient interactive learning.

Another primary challenge in RL is to improve its robustness and generalization to small changes in the environments. A growing number of studies (Pinto et al., 2017; Vinitsky et al., 2020; Ferguson & Law, 2018) show that RL policy is vulnerable to small perturbations. In practice, the perturbations are usually caused by the difference between the training environment and the deployed environment, e.g., changes of the position and size of obstacles/objects. Hence, how to improve the generalization to such changes is critical. When addressing the sparse reward problem, the engineered sub-tasks or relabeled goals might be redundant or too easy to provide effective feedback. Adversarially modifying their environments can produce more diverse and informative experiences for RL. Although modifying environment has been recently studied to assist RL (Co-Reyes et al., 2020; Gur et al., 2021; Wang et al., 2019), it can make long-horizon tasks even more challenging and reward-sparse, hence detrimental to RL's efficiency.

In this paper, we address the above two challenges by automatically generating a curriculum of sub-tasks with adversarial environments adaptive to the learning progress of RL. The curriculum decomposes a long-horizon task into easier sub-tasks offering dense rewards and modifies each sub-task to improve RL policy's tolerance to perturbed environments. As illustrated in Fig. 1, our approach, "environment-adversarial sub-Task curriculum (EAT-C)", generates a tree-structured curriculum by (1) a path-planning policy that recursively decomposes a task (e.g., a state-goal pair (s, g)) as coarse-to-fine sub-task sequences (e.g., consecutive sub-goals between (s, g)) of multi-granularity; and (2) an environment generator (EG) policy that adversarially modifies each sub-task's environment. The path-planner is trained to produce the most cost-efficient/shortest path in each granularity level, while the environment policy is trained to reduce the expected return of the RL agent. In EAT-C, training these two policies does not require external supervision: we instead collect the time costs and rewards of the RL agent on previous sub-tasks to train them towards generating better curricula adaptive to RL progress.

The two curriculum policies and RL can efficiently learn from each other by iterating the above mutual-boosting scheme on the tree-structured curriculum of sub-tasks, which naturally provide an easy-to-hard curriculum to train each policy: (1) the top-down construction of the tree trains the path-planner to first interpolate a few sub-goals between (s, g) and then gradually increases the sub-goals for finer-grained planning; (2) the adversarial training between EG and RL follows a bottom-up traversal of the tree, i.e., it starts from a fine-grained sequence of easier sub-tasks with more frequent perturbations between (s, g), which is easy for both policies, and then progressively changes to more challenging cases, i.e., a coarse sequence of long-horizon sub-tasks with fewer environment changes between (s, g). Experiments on challenging discrete navigation and continuous control tasks demonstrate that EAT-C considerably improves the efficiency of RL and its generalization to perturbed environments. Compared with recent RL methods with hand-crafted curricula, environment generators, hierarchical policies, or path-planning, EAT-C consistently achieve better generalization on different tasks in unseen environments.

2. Related Work

Goal-conditioned RL (Pong et al., 2018; Kaelbling, 1993) aims to train a unified policy for different goals/tasks. It requires extensive exploration and training on different goals but still easily fails to reach distant goals in practice. Goal relabeling and reward shaping (Andrychowicz et al., 2017; Nasiriany et al., 2019) have been developed to mitigate these issues by modifying the rewards to be dense but introduce extra bias and cannot control the utility of modified goals/rewards to the targeted ones. In order to provide tasks at the appropriate level of difficulty for the agent to learn, (Held et al., 2018) and (Racanière et al., 2019) train a goal/task generator based on the agent's performance in previous tasks. But it is usually non-trivial to determine the difficulty level for each training stage. In EAT-C, RL is trained with dense rewards provided by an automatically generated sequence of easier sub-tasks, which are adaptive to RL progress. The adversarial EG further modifies each sub-task to be sufficiently challenging and diverse, which enables more efficient RL.

Planning (Sutton & Barto, 2018; LaValle, 2006) is helpful to solving long-horizon tasks (Levine et al., 2011) by interpolating intermediate sub-goals or searching for the shortest path between two nodes (states) (LaValle, 2006; Dayan & Hinton, 1993; Eysenbach et al., 2019; Howard & Kelly, 2007; Werling et al., 2012), but they require accurate distance metric, which is usually unavailable. Sequentially planning (Schmidhuber & Wahnsiedler, 1993; Zhang et al., 2020) sub-goals from the starting state to the goal is inefficient in complex tasks, as it needs to search in a large space. In RL, planning requires an environment model or learns a value function to improve the policy (Levine et al., 2011; Lau & Kuffner, 2005; Elbanhawi & Simic, 2014), which can be as challenging as model-free RL. In EAT-C, we train a path-planner that "learns to plan" and to directly generate the shortest (in terms of time cost) path for the RL agent. Hence, it requires neither expensive search procedures nor a predefined distance metric. Moreover, it does not rely on learning a world/robot model or a value function.

Several recent works (Wang et al., 2019; Portelas et al., 2019; Gur et al., 2021; Yang et al., 2022) show that the efficiency and generalization of RL can be improved by training the policy in different environments. They (Portelas et al., 2019; Wang et al., 2019) train the RL agent based on a curriculum of diverse environments by sampling the parameters of environment features. However, the generated environments can be infeasible or too challenging for the RL agent and it is non-trivial to control their hardness. Moreover, estimating the distribution of environments requires evaluating the RL policy on many sampled environments, which is costly and inaccurate especially for complicated environments. In EAT-C, we adversarially modify the environment for each sub-task, whose hardness is controlled by both the path planner and EG, so the modified sub-task is still feasible for RL to learn. Moreover, a policy network for EG facilitates the modification of complicated environments and it is efficient to train in our mutual boosting scheme due to the easy-to-hard curriculum.

Hierarchical planning (HP) methods (Nasiriany et al., 2019; Jurgenson et al., 2020; Pertsch et al., 2020) search for a sequence of sub-goal on a tree to guide an agent but building the hierarchical partition of all possible sub-goals can be expensive. For example, (Kaelbling & Lozano-Pérez, 2011) learns to predict sub-tasks of a tree structure based on pre-defined motion primitives. Hierarchical RL (HRL) for goal-reaching tasks has been studied in (Zhang et al., 2021; Nachum et al., 2018). (Shu et al., 2018) trains an RL agent on a human-designed curriculum of tree-structured sub-goals. In contrast, EAT-C trains a planning policy to decompose a hard task into sub-tasks of multiple difficulty levels by only using the RL's time cost data. It requires neither hierarchical partition of the task space nor a predefined set of sub-tasks. Compared to HP and HRL, EAT-C (1) enables a mutual training between the path-planner and RL; (2) has an adversarial EG to train RL in a more challenging environment for each sub-task; and (3) improves the RL efficiency under the guidance of easy-to-hard curricula automatically determined for each stage. We discuss and compare with more related work in Appendix. E.

3. Problem Formulation

3.1. Goal-conditioned Reinforcement Learning

Goal-conditioned RL or multi-goal RL learns a policy that can adapt to different goals. Given the state space $\mathcal{S}$, the action space $\mathcal{A}$, and the goal space $\mathcal{G}$, a goal-conditioned policy is a mapping $\pi(a|s,g) : \mathcal{S} \times \mathcal{G} \mapsto \mathcal{A}$ that outputs an action a (or probabilities $\Pr(a|s,g)$ over actions $a \in \mathcal{A}$) given a state-goal pair (s,g). An RL agent starts from a initial state $s = s_0$ and uses $\pi(a|s,g)$ to take a sequence of actions and interact with the environment, which determines the agent's new state and reward after the action taking in each step. The interaction with the environment is defined by a Markov decision process (MDP) $\{\mathcal{S}, \mathcal{A}, \mathcal{G}, p, r, \gamma\}$, where $p(s'|s,a) \triangleq \Pr(s_{t+1} = s'|s_t = s, a_t = a)$ is the transition probability for the agent from state s to s' after taking action a, $r(s,a|g) : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \mapsto \mathbb{R}$ is a reward function, and $\gamma \in [0,1]$ is a discount factor. For example, it is common to define $r(s,a|g) \triangleq \mathbf{1}\{d(s,g) \leq \epsilon\}$, where $\mathbf{1}$ is the indicator function and $d(\cdot,\cdot)$ is a distance metric, so the agent achieves reward of 1 if reaching an ϵ-neighborhood of the goal g.

The learning objective of goal-conditioned RL is to maximize its expected return over different tasks (s_0, g), i.e., $\max_\pi \mathbb{E}_{(s_0,g)}[\mathbb{E}_\pi(R_0)]$ where the return at step-t is defined as $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_t, a_t|g)$. This is usually used as the objective for policy gradient methods. Define the action-value function $Q(s,a|g) \triangleq \mathbb{E}(R_t|s_t = s, a_t = a, g)$, the optimal policy π^* achieves the maximal $Q(s,a|g)$ for any feasible (s,a,g). Define the value function $V(s|g) \triangleq \mathbb{E}(R_t|s_t = s, g) = \mathbb{E}_{a\sim\pi}[Q(s,a|g)] = \sum_{a\in\mathcal{A}} \pi(a|s,g)Q(s,a|g)$. To reduce the variance of R_t, Actor-critic methods additionally learns a model of V or Q as a "critic" to the "actor" π. Their training objectives aim to minimize the Bellman residual, i.e., the difference between the two sides of Bellman equation:

$$\begin{aligned} Q(s_t, a_t|g) = & r(s_t, a_t|g) + \\ & \gamma \mathbb{E}_{s_{t+1}\sim p}[\mathbb{E}_{a\sim\pi}[Q(s_{t+1}, a|g)]]. \end{aligned} \tag{1}$$

In experiments, to encourage exploration, we use soft actor-critic (SAC) (Haarnoja et al., 2018a).

3.2. Generating Sub-task Curricula for Goal-Conditioned RL

Training goal-conditional RL on long-horizon tasks can be highly inefficient since sparse rewards can only be achieved after taking a long sequence of actions, which cannot provide informative feedback to improve the policy. Moreover, the immature policy in earlier stages can easily fail and make the rewards even more sparse. In this paper, we train **a path-planning policy** π_p to recursively decompose a long-horizon task into coarse-to-fine sequences of easier sub-tasks as a sub-task tree. Solving any of these sequence can accomplish the original task but the finer ones composed of more sub-tasks provide denser rewards to RL. Hence, the path-planner generates an easy-to-hard curriculum to train the RL agent, which starts from learning easier sub-tasks in finer sequences and gradually focus on more challenging sub-tasks in coarser ones. However, some sub-tasks can still be either too trivial for RL or redundant to other sub-tasks. Moreover, the RL agent is not trained to adapt to small changes in the environment. To address these two issues, we propose **an environment generator (EG)** π_e that adversarially modifies the environment of each sub-task to be more challenging and informative for RL. Therefore, the path-planner and the EG are complementary in producing an efficient curriculum of tasks and environments to train the RL agent.

3.2.1. Co-operative Path Planner: Learning Easy-to-Hard Sub-Tasks

We extend "sub-goal tree (SGT)" (Jurgenson et al., 2020) to generate our curriculum of sub-tasks. Given an long horizon task (s_0, g) with initial state s_0 and final goal g, we recursively apply a path planning policy $\pi_p(g|g_i, g_j)$ to interpolate sub-goals between s_0 and g. In particular, given two sub-goals g_i and g_j, sampling from $\pi_p(g|g_i, g_j)$ yields a sub-goal interpolated between g_i and g_j. Hence, we can generate a sub-goal tree $g_{0:T}$ for (s_0, g) by

$$\Pr_{\pi_p}(g_{0:T}|g_0 = s_0, g_T = g) \triangleq \tag{2}$$
$$\Pr_{\pi_p}\left(g_{0:\frac{T}{2}}|g_0, g_{\frac{T}{2}}\right) \Pr_{\pi_p}\left(g_{\frac{T}{2}:T}|g_{\frac{T}{2}}, g_T\right) \pi_p\left(g_{\frac{T}{2}}|s_0, g\right),$$

where $T = 2^K$ with K being the depth of the tree. As shown in Fig. 1, layer-k of the sub-goal tree $g_{0:T}$ interpolate a sequence of $2^k - 1$ sub-goals $g^k_{1:2^k-1} \triangleq \left(g^k_1, g^k_2, \cdots, g^k_{2^k-1}\right)$ between s_0 and g, where $g^k_j = g_{Tj/2^k}$ in $g_{0:T}, \forall j \in \left[2^k - 1\right]$. The goal of path planning is to generate cost-efficient sub-tasks for the RL agent, so we train π_p by minimizing the time cost $c(g^k_{0:2^k})$ of the sub-goal trajectory $g^k_{0:2^k}$ on each layer-k, i.e.,

$$\min_{\pi_p} J_{\pi_p} \triangleq \mathbb{E}_{(s_0,g)}\mathbb{E}_{g_{1:T-1}\sim\pi_p}\left[\sum_{k=1}^{K} c(g^k_{0:2^k})\right],$$
$$c(g^k_{0:2^k}) \triangleq \sum_{t=0}^{2^k-1} c\left(g^k_t, g^k_{t+1}\right), \tag{3}$$

where $c(g^k_t, g^k_{t+1})$ represents the time-cost that the agent taken from g^k_t to g^k_{t+1}.

3.2.2. Adversarial-Environment Generator (EG) applied to each Sub-Task

Given the next sub-task (g^k_t, g^k_{t+1}) in layer-k, EG policy π_e adversarially modifies the environment E^k_{t-1} of previous sub-task (g^k_{t-1}, g^k_t) to be more challenging to the RL agent, i.e., sampling subtask-t's environment $E^k_t \sim \pi_e(E|s^k_t, g)$ where $s^k_t \triangleq (E^k_{t-1}, g^k_t, g^k_{t+1})$ denotes the state of EG at subtask-t. As an adversary to the RL agent, the reward function for EG is defined as $r_e(s^k_t, E^k_t|g) \triangleq -\mathbf{1}\{r(s_t, a_t|g) = 1\}$ where (s_t, a_t) refer to the state-action of the RL agent at the end of subtask-t. Thereby, EG receives the minimum reward -1 when the RL agent successfully finishes subtask-t and otherwise the reward is 0. We can also define an MDP for EG, which mainly differs from the RL agent in that each step corresponds to a sub-task so EG is only allowed to modify the environment at the beginning of each sub-task.

Similar to goal-conditioned RL defined in Sec. 3.1, the learning objective of EG is to maximize its expected return over different tasks (s_0, g), i.e., $\max_{\pi_e} \mathbb{E}_{(s_0,g)}[\mathbb{E}_{\pi_e}(R^e_0)]$ where the return is defined as $R^e_t = \sum_{k=1}^{K}\sum_{i=t}^{2^k} \gamma_e^{i-t} r_e(s^k_t, E^k_t|g)$ with discount factor $\gamma_e \in [0, 1]$. By defining the corresponding value function V_e and action-value function Q_e as in Sec. 3.1, we can apply any RL algorithm to train EG, e.g., we use A2C (Mnih et al., 2016) for the experiments.

4. Environment-Adversarial sub-Task Curriculum

4.1. Auto Curriculum Generation and Mutual-Boosting

In EAT-C, we need to jointly train three policies, i.e., the path-planner π_p and EG policy π_e that generate tree-structured curricula of sub-tasks, and the RL policy π to accomplish the targeted tasks. At the first glance, training three policies can be more difficult than training one RL policy and requires to collect more data via interactions. Moreover, it is challenging to directly train a path-planner generating dense sub-goals and EG can also suffer from sparse rewards on long-horizon tasks. However, EAT-C allows the three policies help each other's training via a mutual boosting mechanism, where each policy is progressively trained on a curriculum of easy-to-hard sub-tasks using dense feedback from other policies on the sub-tasks. By iterating this mutual-boosting on sub-task curricula, EAT-C significantly improves the training efficiency of each policy and results in an RL agent with better generalization to unseen tasks and perturbed environments.

In each episode, we train the path-planner during its "top-down" construction of a sub-task tree: it starts from learning to interpolate a few sub-goals between the given task (s_0, g) and gradually moves to more challenging cases of generating dense sub-goals in bottom layers of the tree. Since it aims to generate the most cost-efficient path of sub-goals for the RL agent, we use the time cost of the RL agent on those sub-tasks in the previous episode as training data, which provide dense rewards to accelerate the training of the path-planner.

Given a constructed sub-task tree, we then train the RL policy and EG policy by a "bottom-up" traversal of the sub-task tree, which naturally forms an easy-to-hard curriculum for each policy. Specifically, both policies firstly learn from dense rewards by finishing easier sub-tasks in bottom layers, where the RL agent only needs to reach a nearby sub-goal and the EG is allowed to frequently modify the environments between (s_0, g). Due to the adversarial training between them, they are not only learning from the environments but also from each other. As moving to the top layers, the RL agent receives less guidance from fewer sub-tasks, while the EG can only change the environment once in each long-horizon sub-task. Thereby, they need to improve their policies learned on easier tasks for more challenging tasks. As a result, the RL agent learns to adapt to different tasks and perturbed environments, while the EG learns to In EAT-C, for each long-horizon task, we iterate the above mutual-boosting training on new sub-task curricula re-generated by the updated path-planner and EG for multiple episodes. This is an imitation of human learning that repeatedly practicing the same complicated task in different ways (e.g., different sub-tasking and perturbed environments). The path-planning and adversarial modification of environments are complementary in constructing a curriculum for more efficient RL: the former decomposes a hard task into easier sub-tasks while the latter modifies them to be sufficiently challenging and diverse so the RL agent can learn different skills with better generalization to unseen tasks or perturbed environments.

4.2. EAT-C Algorithm

Top-Down Planning of Sub-task Curriculum We provide the detailed procedure of the top-down construction of the sub-task curriculum and the update of path-planner π_p in Algorithm 1 (more detailed in Appendix. C). For each layer-k from $k = 0$ to $k = K$, EAT-C firstly updates the planning policy π_p in line 4 by an RL algorithm using the time cost data collected on sub-tasks up to layer-k from the previous episode's bottom-up training (i.e., Algorithm 2), and then recursively generates the sub-tasks on layer-k for the current episode (line 5-8). Hence, the path-planner firstly learns to plan coarse trajectories of fewer sub-goals in top layers and gradually increases the sub-goals to form finer paths that can provide more guidance to RL in bottom layers. Since we always use the latest time cost data from $\mathcal{D}_p$ for training, the planning policy π_p keeps being optimized to generate cost-efficient sub-task trajectories for the latest RL policy π.

Algorithm 1 Top-Down Planning of Sub-task Curriculum

```
1: Input: $(s_0, g)$, $T$, planning policy $\pi_p$ and its training set $\mathcal{D}_p$.
2: Output: tree structured sub-goals $g_{0:T}$, $\pi_p$
3: for $k = 1, 2, \dots, K$ do
4:   Apply an RL algorithm to minimize $J_{\pi_p}$ in Eq. (3) computed on time cost data up to layer-$k$ in $\mathcal{D}_p$;
5:   for $t = 1, \dots, 2^{k-1}$ do
6:     Generate the sub-goal $g_t^k \sim \pi_p(g_t^k | g_{t-1}^{k-1}, g_t^{k-1})$;
7:     Add $g_t^k, g_t^{k-1}$ into the trajectory $g_{0:T}^k$ on layer-$k$;
8:   end for
9: end for
```

Bottom-Up Curriculum for RL and EG Given a sub-task tree generated by Algorithm 1 (more detailed shown in Appendix. C), EAT-C trains RL policy π and EG policy π_e by following a bottom-up traversal of the tree. As shown in Algorithm 2 (more detailed shown in Appendix. C), it starts from learning easier sub-tasks on the bottom layer-K and gradually moves to top layer-0 (line 4-12), which recovers the original long-horizon task. In each layer, we reset the initial state of the RL agent (line 5) and apply π_e and π to a sequence of 2^k sub-tasks $g_{0:2^k-1}^k$ towards the final goal g (line 6-10), and then we update the two policies using the experiences collected on these sub-tasks (line 11). On each sub-task, EG adversarially perturbs the environment (line 7) and the RL agent is then applied to accomplish this modified sub-task (line 8). If the RL agent fails and ends at a state s, EAT-C recursively invoke the planning policy π_p to add more sub-goals between s and the sub-task's goal to provide more detailed guidance to the RL agent until it reaches the goal. This is described in line 9 and line 13-21.

Algorithm 2 Bottom-Up traversal in EAT-C

```
1: Input: RL and EG policy $\pi, \pi_e$, sub-goal tree $g_{0:T}$, $\tau_{\max}$, $\epsilon$
2: Output: $\pi$, $\pi_e$, $\mathcal{D}_p$
3: Initialize: $\pi_p$'s training set $\mathcal{D}_p \leftarrow \emptyset$, RL's replay buffer $\mathcal{D} \leftarrow \emptyset$, EG's replay buffer $\mathcal{D}_e \leftarrow \emptyset$
4: for $k = K, \dots, 1, 0$ do
5:   Reset RL agent's initial state to $g_0$;
6:   for $t = 1, 2, 3, \dots, 2^k$ do
7:     EG modifies the environment $\mathbf{E}_{t-1}^k$ to $\mathbf{E}_t^k$;
8:     Apply RL agent to complete sub-task $(g_{t-1}^k, g_t^k)$, and store $(s_\tau, a_\tau, r(s_\tau, a_\tau | g_t^k), s_{\tau+1})$ to $\mathcal{D}(\tau \leqslant \tau_{\max})$;
9:     REACH$(s, g_t^k)$;
10:  end for
11:  Update $\pi$ and $\pi_e$ using samples in $\mathcal{D}$;
12: end for
13: Procedure REACH$(s, g)$:
14: if $d(s, g) \leq \epsilon$ then
15:   $\mathcal{D}_e \leftarrow \mathcal{D}_e \cup (s_\tau, b_t, r_e(s_\tau, E_t^k | g), g_t^k)$;
16:   $\mathcal{D}_p \leftarrow \mathcal{D}_p \cup (s_0, s_\tau, \tau)$,  $s_0 \leftarrow s_\tau$;
17: else
18:   Re-apply $\pi_p$ to interpolate temporary sub-goals for $(g_{t-1}^k, g_t^k)$;
19:   Repeat Line.8 with new generated sub-goal sequence;
20:   REACH$(s, g)$;
21: end if
```

Algorithm 3 EAT-C

```
1: Input: $p_0$, $T$, $\tau_{\max}$, $\epsilon$, $n$
2: Output: $\pi$, $\pi_p$, $\pi_e$
3: Initialize: $\pi$, $\pi_p$, $\mathcal{D}_p$
4: while not converge do
5:   Sample a task $(s_0, g)$ with $s_0 \sim p_0$ and $g \in \mathcal{G}$;
6:   for episode $= 1, 2, \dots, n$ do
7:     Algorithm 4: top-down construction of a sub-task tree $g_{0:T}$, train planning policy $\pi_p$ based on $\mathcal{D}_p$;
8:     Algorithm 5: bottom-up traversal of the sub-task tree $g_{0:T}$, train RL policy $\pi$ and EG policy $\pi_e$, collect $\mathcal{D}_p$;
9:   end for
10: end while
```

EAT-C algorithm The complete procedure of EAT-C is introduced in Algorithm 3. Given a long-horizon task (s_0, g)

(line 5), EAT-C iterates between the top-down and bottom-up procedures in Algorithm 4-5 for n episodes (line 6-9) before moving to a new long-horizon task. Therefore, the planning policy π_p and EG policy π_e are optimized to produce better curricula of sub-tasks to train the RL agent to complete task (s_0, g) while the RL agent learns skills for solving different sub-tasks and generalizing to non-trivial changes of the environment.

5. Experiments

In this section, we evaluate EAT-C and compare it with a broad range of RL methods on three benchmarks, i.e., navigation and manipulation of a 6-point 2D robot to push an object to a goal state, a 7DoF robotic arm control problem, and three compositional tasks in a discrete space. In these experiments, we mainly focus on their efficiency on long-horizon tasks and generalization to environments with small changes. Moreover, we present an ablation study to evaluate the contribution of each part in EAT-C. In addition, we provide case studies to analyze the planned sub-task tree and the modified environment for each sub-task, which explains why EAT-C can improve RL in several aspects.

2D pusher (Yamada et al., 2020). As shown in Fig. 9, this is a robot navigation and manipulation task in a continues space: a 2D robot with a 4-joint arm needs to firstly navigate to an object and then push it to a goal location within an environment of multiple obstacles. We randomly sample diverse environments and tasks for training and test from a uniform distribution. In 2D pusher, the agent only receives reward when it navigates near the object or pushes the object to the goal state.

Discrete space tasks (Maxime Chevalier-Boisvert & Pal, 2018). We train and test RL policies on three types of compositional tasks depicted in fig 4, i.e., hunting, scavenging, and salad-making, as illustrated in Fig. 4.(a)-(c). The agent needs to take two or three key steps to finish each task and only get reward when finishing each key step. In EAT-C, the path-planner is applied to every two consecutive key steps. EG perturbs the environment by moving objects including tree and stones.

7DoF robotic arm, To demonstrate that EAT-C can adapt to more complex tasks, we conduct an experiment of controlling a 7DoF (degrees of freedom) robotic arm (i.e., the one used in (Jurgenson et al., 2020)) to evaluate how EAT-C performs in a complicated control task. We use MuJoCo as the simulator. In this experiment, the robotic arm learns to avoid obstacles and reach a goal state (as shown in Fig. 2).

More details of the implementation of environments and experiments are introduced in Appendix A.

Baselines: We compare EAT-C with a broad class of RL methods having related ideas to EAT-C: (1) ALP-

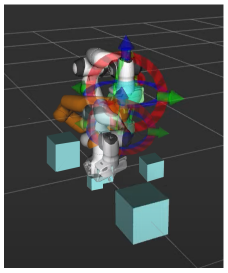


Figure 2: 7DoF Robotic Arm in a training environment with randomly sampled obstacles (those cyan cubes).

GMM (Portelas et al., 2019) that generates a curriculum of diverse tasks with large progress for goal-conditioned RL; (2) POET (Wang et al., 2019) with two auxiliary agents for generating a curriculum of adversarial yet solvable environments to accelerate RL; (3) Ecological RL (Co-Reyes et al., 2020) that dynamically modifies the environment to improve non-episodic (and thus long-horizon) RL without reset of the initial state; (4) A hierarchical RL method (Zhang et al., 2021); (5) (Zhang et al.) trains the RL agent by modelling a goal proposal curriculum that samples goals at the frontier of the set of goals that an agent is able to reach. All these baselines and EAT-C need to invoke an RL algorithm as their subroutine. For fair comparisons, we use Soft Actor Critic (SAC) (Haarnoja et al., 2018b) in all evaluated methods for its stable and promising performance. All methods are not allowed to modify the environments during test since test environments are assumed to be the realistic ones in which we deploy the RL agents. More details of experimental settings are given in Appendix A.

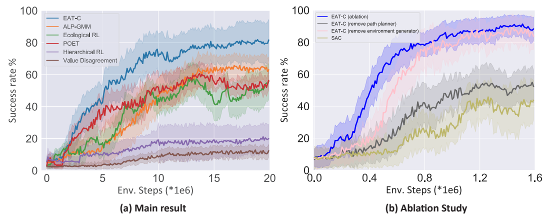


Figure 3: **(a)** report the success rate (mean±std averaged over 6 random seeds) of EAT-C and baselines on test tasks in 2D Pusher environments. **(b)** Ablation study of EAT-C on 2D Pusher tasks.

5.1. Main Results

We report the performance of EAT-C and all baselines evaluated on the test tasks in Fig. 3(a) for 2D pusher, in Fig. 4(d)-(f) for the discrete space tasks and in Table. 1. Due to the limited space, we provide a more clear version of main results in Appendix. D.2. In all experiments, EAT-C outperforms all other baselines by a large margin on both the learning efficiency and the final generalization performance to test tasks in new environments. In most experiments, baselines adopting a curriculum of environments, i.e., ALP-GMM, POET, and Ecological RL, performs worse than EAT-C but better than the other baselines without changing environments for training. This indicates that building a curriculum of training environments is essential to improving RL's generalization and robustness to small changes in the deployed environments. The final performance on 7DoF robotic arm control problem shown in Table. 1 indicates that EAT-C improves the RL policy in complex tasks.

Among the three compositional tasks in Fig. 4, hunting and scavenging contain a moving object, i.e., the deer and the predator, which require the RL agent to adapt to the changes of their locations. On these two tasks, EAT-C exhibits more advantages over other baselines than on the salad-making task, which does not contain any moving object. Therefore, EAT-C enables RL to learn to adapt to changes in the environment efficiently. Our analysis about the main results is more detailed in Appendix. D.3.

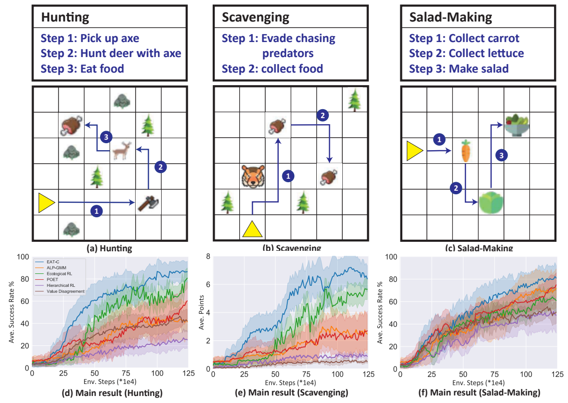


Figure 4: **(a)-(c)** illustrate the 2-3 key steps for completing each task. In Scavenging, the agent will have 2 points when it collects food each time. **(d)-(f)** report different methods' performance (mean±std over 10 random seeds) on multiple test tasks.

Methods	Average Collision Rate	Success Rate
EAT-C	$\mathbf{0.22} \pm 0.05$	$\mathbf{0.873} \pm 0.027$
ALP-GMM	0.34 ± 0.07	0.524 ± 0.092
POET	0.42 ± 0.07	0.544 ± 0.084
SGT-PG	$0.25 \pm \mathbf{0.02}$	$0.772 \pm \mathbf{0.014}$

Table 1: Main results for the experiments on 7DoF robotic arm. In more complex control tasks, EAT-C achieves the best success rate in completing test tasks, and has the lowest possibility of collision.

5.2. Ablation study

EAT-C jointly trains a path-planner and an environment generator (EG) to produce a curriculum of sub-tasks to improve RL. Hence, we conduct two thorough ablation studies of how the performance changes if removing each component from EAT-C under different training/test conditions. In particular, we compare the original EAT-C with three variants, i.e., EAT-C with path-planner removed, EAT-C with EG removed, and SAC (with both removed), on the 2D Pusher tasks. Since the last two variants are not trained on perturbed environments, for fair comparisons, we use the same environment for both training and test and only create new test tasks by sampling (s_0, s_{obj}, g) in Fig. 3(b).

To evaluate the generalization and robustness, which are the advantages due to the adversarial environment generator, we conduct an ablation study that evaluates different methods on multiple new random environments during the test phase and report the test performance in Table. 2.

Test Setting	Multiple New Random Environments	Training Environment
EAT-C	80.24 ± 12.25	92.04 ± 6.49
EAT-C (no EG)	42.23 ± 10.34	85.47 ± 9.12
EAT-C (no Planner)	27.58 ± 14.67	46.02 ± 10.3
SAC	20.83 ± 7.24	39.62 ± 12.25
Hierarchical RL	22.04 ± 10.44	68.27 ± 6.99

Table 2: Success rate on test tasks in random environments and training environment. Different from the ablation study in Fig. 3(b), we evaluate EAT-C and baseline methods on multiple new random environments during the test phase. The results demonstrate that EG in EAT-C can improve the generalization and robustness of RL policy.

When the path-planner is removed from EAT-C, we no longer have any easy-to-hard curriculum of sub-tasks to train the RL agent or EG and they can only learn inefficiently from the original long-horizon tasks. The adversarial environments generated by EG make tasks for RL even harder and unsolvable. Hence, we can observe that, in Fig.3(b), it only completed 50% of the test tasks within 1.6×10^6 environment steps, compared to 90% of the original EAT-C. This phenomenon becomes more obvious when evaluating on random environments (27.58 vs. 80.24 in Table. 2). This indicates that the plan-planner and its generated sub-task tree are critical in creating an effective curriculum for RL.

When EG is removed from EAT-C, we still have the curriculum of sub-tasks to train the RL agent, but some sub-tasks might be too trivial or redundant to provide informative feedback for improving the RL agent. This results in poorer efficiency in earlier-stages compared to the original EAT-C with an EG: in Fig. 3(b), it reaches 80% success rate after 0.84×10^6 interaction steps instead of 0.73×10^6 steps required by the original EAT-C. Although ETA-C without EG can eventually achieve a comparable success rate at the end of training, the learned RL policy cannot generalize to diverse environments (42.23 vs. 80.24, in Table. 2). During later stages, the success rate fluctuates unstably over time, while EAT-C with an EG performs more robustly due to the adversarial environments used for training. Moreover, EAT-C significantly improves SAC by a large margin via running SAC on an automatically generated curriculum of sub-tasks, which implies the importance of curriculum on improving RL's efficiency. More discuss and analysis about ablation study results is in Appendix. D.3.

Due to the limited space, we report **the results of more ablation studies** about EG in EAT-C and curriculum RL methods of baseline in Appendix. D.4 to Appendix. D.7.

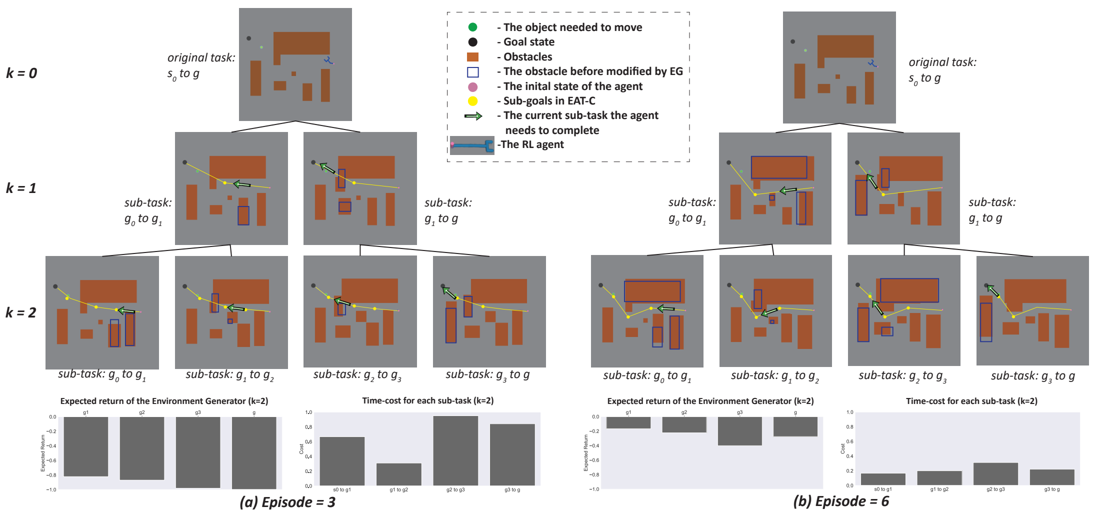


Figure 5: Visualization of EAT-C. A 2D robot with a 4-joint arm starts from the initial state (pink), navigates to the object (green) location, and then pushes the object to the goal state (black). The histograms in (a) and (b) represents the expected return of EG taking action b_t, and the costs of sub-tasks predicted by the path planner in layer $k = 2$, respectively.

5.3. How does EAT-C work? An Empirical Study

To better understand how the path planner and EG help RL in EAT-C, in Fig. 9, we visualize the sub-task tree (with layer $k \in \{0, 1, 2\}$) generated by the path-planner and the EG's modifications to the environment in each sub-task at Episode 3 and Episode 6 (episode was defined in Alg. 3) for a 2D Pusher task (s_0, s_{obj}, g). In the histograms, we also report the expected return of EG and the time cost of the RL agent on each sub-task from the bottom layer $k = 2$ for the two episodes.

The curriculum of sub-tasks generated by the path-planner. Each plot on the tree describes a sub-task, where the arrow highlights the sub-task and the yellow trajectory denotes the sequence of all sub-tasks of the layer. Comparing the sub-tasks in different layers, bottom layers (e.g., $k = 2$) provides more guidance and dense rewards to the RL agent while the sub-tasks in upper layers (e.g., $k = 1$) are much harder. Comparing the same-layer sub-tasks generated in different episodes, the sub-tasks in Episode 3 do not take all obstacles into account, e.g., some sub-task sequences trespass obstacles and some sub-tasks are too close to obstacles, because the planning policy is not fully optimized yet to produce a cost-efficient path for the RL agent. Hence, the time costs for the RL agent to accomplish these sub-tasks can be much higher than later episodes. Moreover, the time costs of some sub-tasks can be much higher than others and thus cannot provide dense rewards to assist RL. On the contrary, in Episode 6, the path-planner has learned to generate cost-efficient sub-tasks with similar hardness so the trajectories and sub-goals are distant from the obstacles and can provide dense rewards facilitating RL. Comparing the histograms of time costs for the two episodes, the RL agent is significantly improved by learning to complete the sub-tasks in the easy-to-hard (bottom-up) curriculum.

Adversarial modifications to obstacles in the environments: In each sub-task plot of Fig. 9, EG adversarially modifies some obstacles by changing their previous sizes and positions (depicted by the blue boxes) to make the sub-tasks sufficiently challenging and diverse. Similar to the path-planner, EG is improved over time: for example, in the sub-task "g_0 to g_1" on layer-2, the RL agent needs to pass a corridor formed by three obstacles, while EG makes the corridor longer and narrower and thus more challenging for the agent to pass in Episode 6, its modification in Episode 3 is not ideal and even moves one obstacle away from the agent. The improvement of EG is also reflected by its increasing expected return shown in the two histograms. By modifying the environments to be more difficult in sub-tasks, EG encourages the RL agent to learn diverse skills in different sub-tasks. Hence, the sub-tasks are easy for the agent to collect dense rewards but they are non-trivial and informative because of EG's modifications.

6. Conclusion

We propose a mutual learning and auto-curriculum framework "EAT-C" to improve the efficiency of RL on long-horizon tasks as well as its generalization and robustness to new environments. EAT-C trains a planner to decompose a hard task into coarse-to-fine sequences of sub-tasks providing an easy-to-hard curriculum to train an RL agent, while an adversarial environment generator modifies these sub-tasks to be diverse and more informative to learn. The three policies are trained with data collected by each other. On three types of tasks, EAT-C outperforms a diverse set of baselines, e.g., curriculum-based RL, hierarchical RL, and planning-based methods.

References

Andrychowicz, M., Crow, D., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. Hindsight experience replay. In *NIPS*, 2017.

Ao, S., Zhou, T., Long, G., Lu, Q., Zhu, L., and Jiang, J. Co-pilot: Collaborative planning and reinforcement learning on sub-task curriculum. In *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc., 2021.

Co-Reyes, J. D., Sanjeev, S., Berseth, G., Gupta, A., and Levine, S. Ecological reinforcement learning. *ArXiv*, abs/2006.12478, 2020.

Dayan, P. and Hinton, G. E. Feudal reinforcement learning. In Hanson, S., Cowan, J., and Giles, C. (eds.), *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1993.

Elbanhawi, M. and Simic, M. Sampling-based robot motion planning: A review. *IEEE Access*, 2:56–77, 2014. doi: 10.1109/ACCESS.2014.2302442.

Eysenbach, B., Salakhutdinov, R., and Levine, S. Search on the replay buffer: Bridging planning and reinforcement learning. In *NeurIPS*, 2019.

Fang, M., Zhou, T., Du, Y., Han, L., and Zhang, Z. Curriculum-guided hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Ferguson, M. and Law, K. Learning robust and adaptive real-world continuous control using simulation and transfer learning. *ArXiv*, abs/1802.04520, 2018.

Florensa, C., Duan, Y., and Abbeel, P. Stochastic neural networks for hierarchical reinforcement learning. In *ICLR (Poster)*.

Francis, A., Faust, A., Chiang, H.-T. L., Hsu, J., Kew, J. C., Fiser, M., and Lee, T.-W. E. Long-range indoor navigation with prm-rl. *IEEE Transactions on Robotics*, 36: 1115–1134, 2020.

Gur, I., Jaques, N., Malta, K., Tiwari, M., Lee, H., and Faust, A. Adversarial environment generation for learning to navigate the web. *ArXiv*, abs/2103.01991, 2021.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018a.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. Soft actor-critic algorithms and applications. *ArXiv*, abs/1812.05905, 2018b.

Held, D., Geng, X., Florensa, C., and Abbeel, P. Automatic goal generation for reinforcement learning agents. *ArXiv*, abs/1705.06366, 2018.

Howard, T. M. and Kelly, A. Optimal rough terrain trajectory generation for wheeled mobile robots. *The International Journal of Robotics Research*, 26:141 – 166, 2007.

Jurgenson, T., Avner, O., Groshev, E., and Tamar, A. Sub-goal trees - a framework for goal-based reinforcement learning. *ArXiv*, abs/2002.12361, 2020.

Kaelbling, L. P. Learning to achieve goals. In *IN PROC. OF IJCAI-93*, pp. 1094–1098. Morgan Kaufmann, 1993.

Kaelbling, L. P. and Lozano-Pérez, T. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pp. 1470–1477, 2011. doi: 10.1109/ICRA.2011.5980391.

Lau, M. and Kuffner, J. J. Behavior planning for character animation. In *Symposium on Computer Animation*, pp. 271–280. ACM, 2005.

Laud, A. D. *Theory and Application of Reward Shaping in Reinforcement Learning*. PhD thesis, USA, 2004.

LaValle, S. M. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

Levine, S., Lee, Y., Koltun, V., and Popović, Z. Space-time planning with parameterized locomotion controllers. 30 (3), 2011.

Lillicrap, T., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2016.

Maxime Chevalier-Boisvert, L. W. and Pal, S. Minimalistic gridworld environment for openai gym. 2018.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, February 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

Nachum, O., Gu, S. S., Lee, H., and Levine, S. Data-efficient hierarchical reinforcement learning. In *NeurIPS*, 2018.

Nasiriany, S., Pong, V. H., Lin, S., and Levine, S. Planning with goal-conditioned policies. In *NeurIPS*, 2019.

Pertsch, K., Rybkin, O., Ebert, F., Finn, C., Jayaraman, D., and Levine, S. Long-horizon visual planning with goal-conditioned hierarchical predictors. *ArXiv*, abs/2006.13205, 2020.

Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. Robust adversarial reinforcement learning. In *ICML*, 2017.

Pong, V. H., Gu, S., Dalal, M., and Levine, S. Temporal difference models: Model-free deep rl for model-based control. *ArXiv*, abs/1802.09081, 2018.

Portelas, R., Colas, C., Hofmann, K., and Oudeyer, P.-Y. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. In *CoRL*, 2019.

Racanière, S., Lampinen, A., Santoro, A., Reichert, D. P., Firoiu, V., and Lillicrap, T. Automated curricula through setter-solver interactions. *ArXiv*, abs/1909.12892, 2019.

Schmidhuber, J. and Wahnsiedler, R. Planning simple trajectories using neural subgoal generators. In *Proceedings of the Second International Conference on From Animals to Animats 2: Simulation of Adaptive Behavior: Simulation of Adaptive Behavior*, pp. 196–202, 1993. ISBN 0262631490.

Shu, T., Xiong, C., and Socher, R. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. *ArXiv*, abs/1712.07294, 2018.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Vinitsky, E., Du, Y., Parvate, K., Jang, K., Abbeel, P., and Bayen, A. Robust reinforcement learning using adversarial populations. *ArXiv*, abs/2008.01825, 2020.

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *ArXiv*, abs/1901.01753, 2019.

Werling, M., Kammel, S., Ziegler, J., and Gröll, L. Optimal trajectories for time-critical street scenarios using discretized terminal manifolds. *The International Journal of Robotics Research*, 31:346 – 359, 2012.

Yamada, J., Lee, Y., Salhotra, G., Pertsch, K., Pflueger, M., Sukhatme, G. S., Lim, J. J., and Englert, P. Motion planner augmented reinforcement learning for robot manipulation in obstructed environments. *ArXiv*, abs/2010.11940, 2020.

Yang, Y., Jiang, J., Zhou, T., Ma, J., and Shi, Y. Pareto policy pool for model-based offline reinforcement learning. In *International Conference on Learning Representations*, 2022.

Yingjun, P. and Xin-wen, H. Learning representations in reinforcement learning:an information bottleneck approach. 2019.

Zhang, J., Yu, H., and Xu, W. Hierarchical reinforcement learning by discovering intrinsic options. *ArXiv*, abs/2101.06521, 2021.

Zhang, T., Guo, S., Tan, T., Hu, X., and Chen, F. Generating adjacency-constrained subgoals in hierarchical reinforcement learning. *ArXiv*, abs/2006.11485, 2020.

Zhang, Y., Abbeel, P., and Pinto, L. Automatic curriculum learning through value disagreement. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *NeurIPS 2020*. Curran Associates, Inc.

A. Details of Environment Implementations

A.1. 2D pusher

The positions of the robot, object and goal are defined as p_{rob}, p_{obj}, and g, respectively, and T is the maximum number of episodes (i.e., the horizon). We train EAT-C on 50 environments with the positions and sizes of obstacles randomly sampled from uniform distributions. In each environment, we randomly sample 80 training tasks with different (s_0, p_{obj}, g), where s_0 is the initial state, p_{obj} is the state of the object, and g is the goal state. Most evaluated policies in our experiments need ≥ 250 steps to finish each task so they are long-horizon tasks. The rewards are sparse since the agent only receives a reward when near the object or when pushing the object and reaching the goal. In EAT-C, the path-planner generates sub-tasks between (s_0, p_{obj}) and (p_{obj}, g). For every sub-task, EG can perturb the sizes and locations of at most three obstacles within pre-defined ranges. For the test, we randomly sample 30 new environments each having one randomly sampled task. We simulate the environment of 2D pusher in Mujoco physics engine (Todorov et al., 2012). An 2D-pusher agent with four joints can take actions of six dimensions, two for navigation and the rest four for arm control. The map size is 1×1 so both the x and y coordinates lie in $(-0.5, 0.5)$. The x and y coordinates of goals and objects are randomly sampled from uniform distributions of $\mathcal{U}(-0.35, -0.2)$ and $\mathcal{U}(-0.15, 0.1)$, respectively. The initial state of the agent is randomly sampled from uniform distribution of $\mathcal{U}(-0.05, 0.3)$. For obstacles, their initial coordinates and sizes are randomly drawn from an uniform distribution, as explained in the first row of Table. 3. We train 2D pusher using sparse reward: when the robot reaches a vicinity of the object or the sub-goal state ($||p_{rob} - p_{obj}||_2 \leqslant 0.05$) the agent will receive a reward$= 150/2^k$, where k is the layer of the sub-task tree where the sub-goal is located. Once the agent pushes the object to the goal state with $||p_{goal} - p_{obj}||_2 \leqslant 0.05$, the agent will receive a one-time reward$= 150$; otherwise there is no reward. By taking an action, EG can change the size and the location of $0 \sim 3$ obstacles near the agent. Assume that there are n obstacles in the environment, and we represent each obstacle-i by its location (x_i, y_i) (2D coordinates) and size (w_i, h_i) (width and height) as $\theta_i = (x_i, y_i, w_i, h_i)$. The action b_t of EG is defined as

$$b_t \triangleq \Delta\theta_i = (\Delta x_i, \Delta y_i, \Delta w_i, \Delta h_i), \ \ \forall i \in [n]. \tag{4}$$

In order to provide feasible and smoothly changing environments to RL along the sub-task trajectory in each layer, and to prevent the environment generator from being too powerful and overly adversarial, it is important to restrict EG from changing the environment too much at one time for each sub-task. Hence, in the experiments, we constrain every dimension in an action of EG not to exceed some threshold, e.g., for x_i, we apply

$$\Delta x_i \leftarrow \min\{\max\{\Delta x_i, \beta_x \cdot x_{\min}\}, \beta_x \cdot x_{\max}\}, \tag{5}$$

where $\beta_x \in [0, 1]$ and $(x_{\min}, x_{\max})$ is the valid range of x-coordinate. The above environment parameterization can be easily extended to other environments so EAT-C is a general and principal scheme that can be adapted to different environments.

A.2. Discrete space tasks

The environment for the three tasks is an $N \times N$ grid. There are 250 environments used for training and 100 environments for test, each associated with one task. It is partially observed by the RL agent: the agent at each state obtains a local egocentric view of a 5×5 grid around it, where the object in each cell of the grid is represented by a C-dimensional one-hot vector (there are C possible types of objects). The agent can pick up and carry one object at a time. It can also combine two objects to construct a new one by putting a carried object onto an existing object, e.g., it can combine wood with metal to make an axe. The RL agent can take action such as moving in the cardinal directions, picking up an object, and dropping an object. In discrete space tasks, the environment generator (EG) can modify the environment by taking an action to move an object/obstacle. In order to provide feasible and smoothly changing environments to RL along the sub-task trajectory, and to prevent EG from being too powerful and overly adversarial, each action of EG can only move every object/obstacle to an adjacent cell around it.

A.3. 7DoF Robotic Arm

In 7DoF robotic arm, both the training and test tasks have 5 obstacles with different and randomly sampled location and size parameters. The start-goal pair of each task are also randomly sampled. Both EAT-C and curriculum RL methods (compared baseline methods) are able to modify exactly the same parameters defining the location and size of each obstacle. We report the success rate of reaching the goal state without collision and the collision rate as the two metrics to evaluate EAT-C and all the baselines. After training, we evaluate them on 100 new random tasks different from the training tasks.

A.4. Environment Encoding of Baselines

The baselines in our experiments include two curriculum RL methods, i.e., ALP-GMM and POET, which both can control some environment-dependent parameters for mutation and generation of the training environments. For their fair comparisons to EAT-C, we set their environment-dependent parameters exactly the same as the ones controlled by EG in EAT-C. For 2D-pusher, as detailed in Table 3, the baselines control the same four environment-dependent parameters as EAT-C. Each parameter is initialized by sampling from a uniform distribution and each mutation step can modify it by a small value if it is within the valid range. For the discrete space tasks, ALP-GMM/POET can generate environments by sampling/mutating the location of each obstacle/object, which is the same parameter controlled by EAT-C. The valid range for location for each obstacle/object is $(0, 1)$ and the mutation step size is 0.1.

Parameter Type	Obstacle x-coordinate x_i	Obstacle y-coordinate y_i	Obstacle width w_i	Obstacle height h_i
Initial Range	$(-0.3, 0.3)$	$(-0.3, 0.3)$	$(0, 0.1)$	$(0, 0.1)$
Mutation Step	$(0.02, 0.02)$	$(0.02, 0.02)$	$(0.05, 0.05)$	$(0.05, 0.05)$
Minimal Value	$(-0.5, -0.5)$	$(-0.5, -0.5)$	$(0, 0)$	$(0, 0)$
Maximal Value	$(0.5, 0.5)$	$(0.5, 0.5)$	$(0.3, 0.3)$	$(0.3, 0.3)$

Table 3: Environment-dependent parameters in baselines on 2D-pusher. Each baseline generates an environment of obstacles by uniformly sampling the four parameters defining each obstacle from the corresponding ranges. It starts from the initial ranges below and can take a mutation step one time to change the lower and upper bounds of each parameter's range, if the two bounds do not exceed their minimal and maximal values listed below. The two numbers in each tuple $(\cdot, \cdot)$ below corresponds to the lower and upper bound of the range.

A.5. Implement of Hyperparameters in Baselines of Curriculum RL methods

The learning efficiency and final performance of curriculum RL methods are usually sensitive to the setting of curriculum related hyperparameters. Therefore, we have finetuned the hyperparameters related to the task/environment propoerties in our experiments, and we keep the rest same to the original paper/code.

POET (Wang et al., 2019): For most hyperparameters, we follow the same setting in the original code of POET (https://github.com/uber-research/poet). We list the tuned hyperparameters in the table below:

Hyperparameter	Tuning set	Final picked value
Max num envs	$\{20, 30, 40\}$	40
Max children	$\{6, 8, 10\}$	8
Reproduction threshould	$\{70, 80, 90, 100\}$	80
Nearest neighbor k	$[4, 8]$	6
Lower criteria	$\{5, 10, 15, 20, 25\}$	10
Higher cirteria	$\{125, 130, 135, 140, 145, 150\}$	130
$\mathcal{N}_{devi}$	$[0.05, 0.15]$	0.1
$\mathcal{N}_{\text{transfer}}$	$\{25, 30\}$	25

Table 4: Finetuned hyperparameters of POET

A complete grid search over all the listed hyperparameters is too costly. Hence, we partitioned these hyperparameters into four groups of strongly related ones (listed below) and applied grid search within each group. For Group 2-4, we started from the default values used in the original POET paper. Since our environments are different, we cannot start from the default hyperparameters for Group 1 used in POET paper. Therefore, we tuned the hyperparameters from Group 1 to Group 4 in a greedy manner. We also tried different orders for Group 2-4. This resulted in hundreds of combinations of hyperparameters and we ran 0.8 millions environment training steps for each combination. We then chose the best combination with the highest success rate for all experiments.

The partition of hyperparameters in POET:

- Group 1: The criteria for mutating environments: {lower criteria, higher criteria, reproduction threshould};

- Group 2: Pool size of environments/ mutation environments: {max num envs, max children};
- Group 3: Exploration and exploitation: $\{\mathcal{N}_{transfer}, \mathcal{N}_{devi}\}$;
- Group 4: Novelty bonus of mutating: {nearest neighbor k}.

ALP-GMM (Portelas et al., 2019): We carefully tuned the hyperparameters of ALP-GMM in the new setting by grid search and chose the one achieving the best performance. Specifically, for the hyperparameters in Algorithm 1 of ALP-GMM paper (https://arxiv.org/pdf/1910.07224.pdf):

- We tuned the probability of random sampling $p_{rnd} \in [0, 05, 0.5]$ and the best one selected is $p_{rnd} = 0.25$;
- We tuned $k_{max} \in \{4, 6, 8, 10, 12\}$ and the number of Gaussians is adapted online by fitting multiple GMMs (here having from 4 to 12 Gaussians);
- we tuned the fitting rate $\mathcal{N} \in \{200, 250\}$ and the best one is $\mathcal{N} = 250$, the same as ALP-GMM paper.

B. Model Architecture and Hyperparameters of SAC (RL Algorithm used in All Experiments)

We use the same neural network architecture (i.e., an MLP) for the RL agent and the same RL algorithm (SAC) in all the experiments of all the methods.

Besides the reward of completing a task/sub-task, it is common in MuJoCo and many other simulators to also issue a small instantaneous reward after taking any action in order to encourage exploration. Moreover, different methods usually need to re-scale this exploration reward because they may need different levels of exploration. In our experiments, we tune the re-scaling factor for every method to get its best performance. Specifically, we chose 0.3 for EAT-C/ALP-GMM/ POET and 8.0 for hierarchical RL/value disagreement/Ecological RL. An explanation of applying a smaller factor for the former three methods is that they already have some strong exploration strategies and a larger factor might downweigh the task reward and thus results in performance degeneration.

Moreover, we use the same coefficient α of the entropy term in SAC's objective for all methods (they all use SAC as the RL algorithm). The coefficient α controls the degree of exploration and is automatically tuned. A complete list of hyperparameters for SAC in 2D-pusher tasks is given in Table 5. They are exactly the same hyperparameters defined in in SAC paper (Haarnoja et al., 2018b) and in Table 1 of their Appendix D except that we choose different values for them in 2D-pusher.

In the discrete space tasks, the environment is a 10×10 grid and the 5×5 partial observation (as mentioned in A.2) of the RL agent can be represented as a $5 \times 5 \times C$ one-hot tensor. We flatten this tensor to a vector and process it by an MLP with three hidden-layers whose output dimensions are $(64, 64, 32)$, respectively. We apply another MLP with three layers of output dimensions $(16, 16, 16)$ to process the inventory observation. The two MLPs' outputs are then concatenated and processed by an MLP with two hidden layers of output dimensions (16 , action_dimension) that outputs a probability distribution over all possible actions. We use ReLU as our nonlinear activation functions in all MLP models except their last layer, which uses a softmax function to compute the probability of taking each action. In EAT-C, the RL agent and EG share the same observations as well as the first two MLP models but they use different MLP models to output the actions. A complete list of hyperparameters of SAC in the discrete space tasks is given in Table. 6.

C. Pseudo-Code of EAT-C

In the training phase, we first randomly initialize π_p, and apply it to predict a sub-task tree using Euclidean distance as an initialization of the cost in line 4 of Algorithm 4 when no time cost data have ever been collected at the very beginning. Given the sub-task tree, we can train the RL agent by a bottom-up curriculum of the sub-tasks on the tree. In particular, we start from the bottom layer and train the RL agent to consecutively complete a sequence of sub-tasks from the starting state to the goal state. As training proceed, the RL agent collects data of the time cost for completing feasible sub-tasks (g, g'). When the RL agent cannot complete the pre-assigned sub-task (g_i^k, g_{i+1}^k) within a time limit, we re-apply π_p to interpolate more sub-goals between (g_i^k, g_{i+1}^k) by line 25 in Algorithm 5. More technical details are given in Algorithm 5. In the test phase, we apply π_p to produce a sub-task tree and only use the sub-task sequence in the bottom layer to guide the RL agent.

Table 5: SAC hyperparameters in EAT-C (2D-pusher)

Parameter	Value
Optimizer	Adam
Learning rate	3.0×10^{-4}
Discount factor (γ)	0.99
Replay buffer size	1.0×10^{6}
Number of hidden layers for all networks	2
Number of hidden units for all networks	400
Minibatch size	256
Nonlinearity	ReLU
Target smoothing coefficient (τ)	5.0×10^{-3}
Target update interval	1
Network update per environment step	1
Entropy target	$-\dim(\mathcal{A})$

Table 6: SAC hyperparameters in EAT-C (discrete space tasks)

Parameter	Value
Optimizer	Adam
Learning rate	5.0×10^{-4}
Discount factor (γ)	0.99
Replay buffer size	1.0×10^{6}
Number of hidden layers for all networks	3
Number of hidden units for all networks	256
Minibatch size	256
Nonlinearity	ReLU
Target smoothing coefficient (τ)	5.0×10^{-3}
Target update interval	1
Network update per environment step	1
Entropy target	$-\dim(\mathcal{A})$

Algorithm 4 Top-Down Planning of Sub-task Curriculum

```
1: **Input:** $(s_0, g)$, $T$, planning policy $\pi_p$ and its training set $\mathcal{D}_p$.
2: **Output:** tree structured sub-goals $g_{0:T}$, $\pi_p$
3: **for** $k = 1, 2, \ldots, K$ **do**
4:   Apply an RL algorithm to minimize $J_{\pi_p}$ in Eq. (3) computed on time cost data up to layer-$k$ in $\mathcal{D}_p$;
5:   **for** $t = 1, \ldots, 2^{k-1}$ **do**
6:     Generate the sub-goal $g_t^k \sim \pi_p(g_t^k | g_{t-1}^{k-1}, g_t^{k-1})$;
7:     Add $g_t^k, g_t^{k-1}$ into the trajectory $g_{0:T}^k$ on layer-$k$;
8:   **end for**
9: **end for**
```

Algorithm 5 More detailed Bottom-Up traversal in EAT-C

```
1: **Input:** RL policy $\pi$, EG policy $\pi_e$, sub-goal tree $g_{0:T}$, $\tau_{\max}$, $\epsilon$
2: **Output:** $\pi$, $\pi_e$, $\mathcal{D}_p$
3: **Initialize:** $\pi_p$'s training set $\mathcal{D}_p \leftarrow \emptyset$, RL's replay buffer $\mathcal{D} \leftarrow \emptyset$, EG's replay buffer $\mathcal{D}_e \leftarrow \emptyset$
4: **for** $k = K, \ldots, 1, 0$ **do**
5:     Reset RL agent's initial state to $g_0$;
6:     **for** $t = 1, 2, 3, \ldots, 2^k$ **do**
7:         EG adversarially modifies the environment $\mathbf{E}_{t-1}^k$ to $\mathbf{E}_t^k$;
8:         **while** $\tau \leq \tau_{\max}$ or $s_\tau \notin B(g_t^k, \epsilon)$ **do**
9:             RL agent takes action $a_\tau \sim \pi(a_\tau | s_\tau, g_t^k)$;
10:            RL agent moves to $s_{\tau+1} \sim p(s_{\tau+1} | s_\tau, a_\tau)$ and receives reward $r(s_\tau, a_\tau | g_t^k)$;
11:            $\mathcal{D} \leftarrow \mathcal{D} \cup (s_\tau, a_\tau, r(s_\tau, a_\tau | g_t^k), s_{\tau+1})$;
12:        **end while**
13:        REACH($s, g_{t+1}^k$);
14:    **end for**
15:    **for** every gradient step **do**
16:        Apply gradient steps in SAC: update $Q,\ V,\ \pi$ using samples drawn from $\mathcal{D}$;
17:        Apply gradient steps in A2C: update $Q_{\pi_e},\ \pi_e$ using samples drawn from $\mathcal{D}_e$;
18:    **end for**
19: **end for**
20: **Procedure** REACH($s, g$):
21: **if** $d(s, g) \leq \epsilon$ **then**
22:     $\mathcal{D}_e \leftarrow \mathcal{D}_e \cup (s_\tau, b_t, r_e(s_\tau, E_t^k | g), g_t^k)$;
23:     $\mathcal{D}_p \leftarrow \mathcal{D}_p \cup (s_0, s_\tau, \tau),\ \ s_0 \leftarrow s_\tau$;
24: **else**
25:     Re-apply $\pi_p^i(g_{t-1}^k . g_t^k)$ to predict temporary sub-goals $g'_{1:n}$for $(g_{t-1}^k, g_t^k)$;
26:     Add $g'_{1:n}$ into the planned sub-goals trajectory $g_{0:2^{k-1}}^k$;
27:     Re-apply agent start from $s_0$ with the new sub-goal trajectory to reach $g_{2^{k-1}}^k$ and collect training data (Follow line.8
        to line.14);
28:     REACH($s, g$);
29: **end if**
```

D. Larger Versions of Figures and Additional Experiment Results

D.1. Larger Version of Main Structure in EAT-C

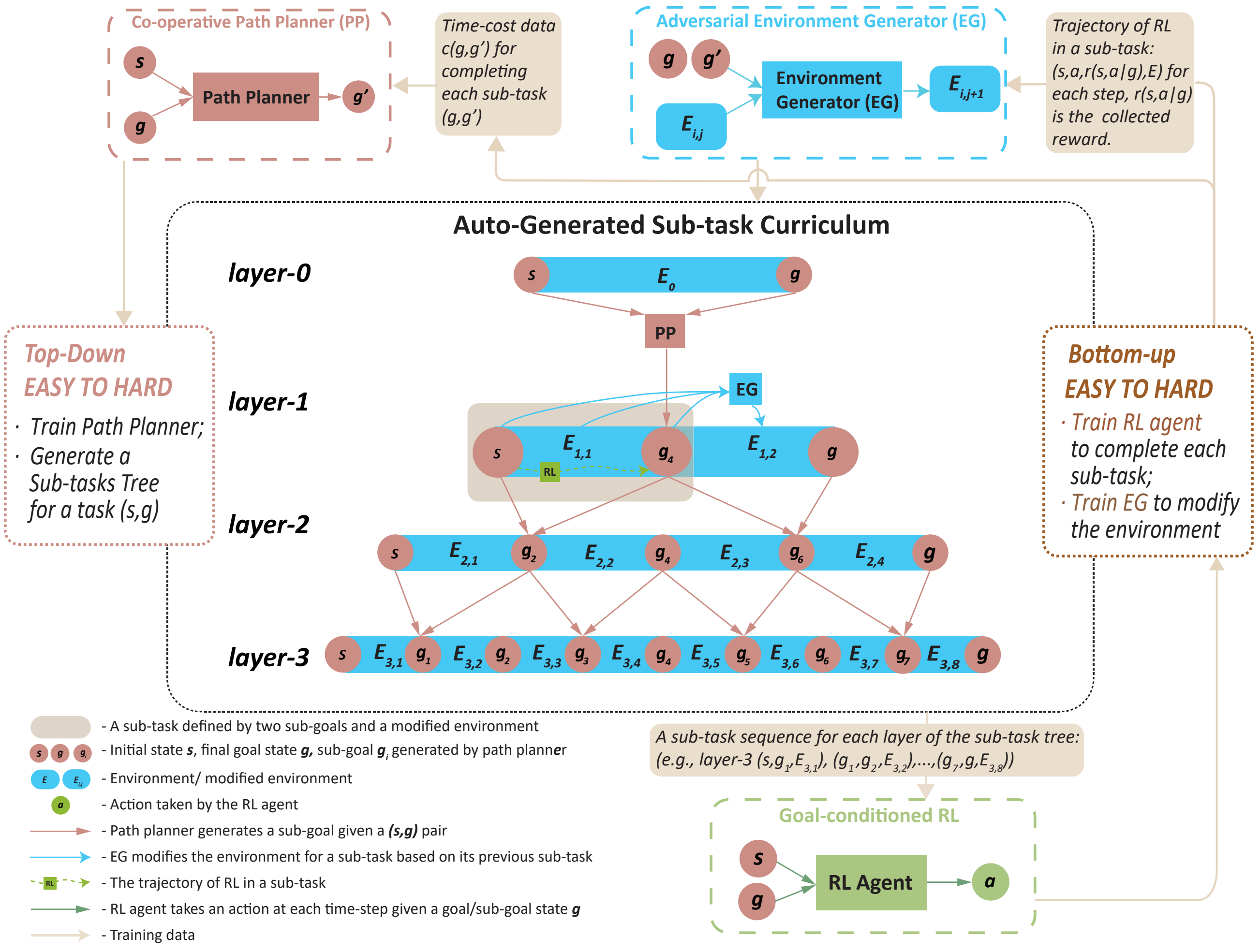

Co-operative Path Planner (PP)
Path Planner
Time-cost data c(g,g') for completing each sub-task (g,g')
Adversarial Environment Generator (EG)
Environment Generator (EG)
Trajectory of RL in a sub-task: (s,a,r(s,a|g),E) for each step, r(s,a|g) is the collected reward.
Auto-Generated Sub-task Curriculum
layer-0
layer-1
layer-2
layer-3
Top-Down EASY TO HARD
· Train Path Planner;
· Generate a Sub-tasks Tree for a task (s,g)
Bottom-up EASY TO HARD
· Train RL agent to complete each sub-task;
· Train EG to modify the environment
A sub-task sequence for each layer of the sub-task tree: (e.g., layer-3 (s,g1,E3,1), (g1,g2,E3,2),...,(g7,g,E3,8))
Goal-conditioned RL
RL Agent
- A sub-task defined by two sub-goals and a modified environment
- Initial state s, final goal state g, sub-goal gi generated by path planner
- Environment/ modified environment
- Action taken by the RL agent
- Path planner generates a sub-goal given a (s,g) pair
- EG modifies the environment for a sub-task based on its previous sub-task
- The trajectory of RL in a sub-task
- RL agent takes an action at each time-step given a goal/sub-goal state g
- Training data

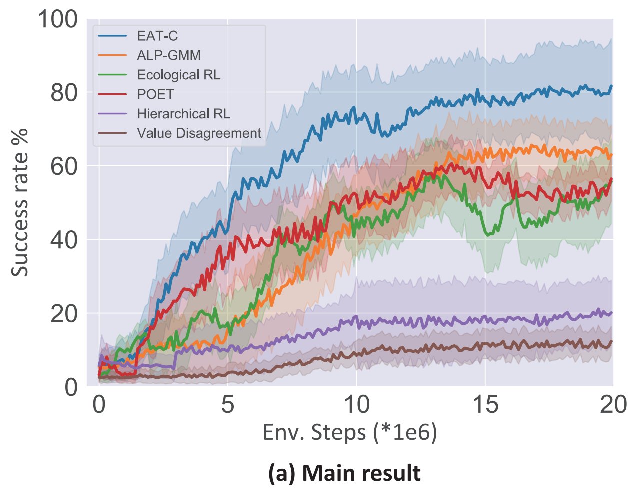
Figure 6: **Main structure of EAT-C:** The path-planner recursively generates a sub-task tree for a task (g_0, g), while the environment generator (EG) adversarially modifies the environment of each sub-task. RL agent is trained on a bottom-up curriculum and its collected data are used to train the path-planner and EG.

D.2. Larger Versions of Plots for Main Results

Considering that the figures and tables of the experiment results in the main paper might be too small to read, we list the main results with a more clear vision (Fig. 7, Fig. 8 and Table. 7).

Methods	Average Collision Rate	Success Rate
EAT-C	$\mathbf{0.22} \pm 0.05$	$\mathbf{0.873} \pm 0.027$
ALP-GMM	0.34 ± 0.07	0.524 ± 0.092
POET	0.42 ± 0.07	0.544 ± 0.084
SGT-PG	$0.25 \pm \mathbf{0.02}$	$0.772 \pm \mathbf{0.014}$

Table 7: Test performance of the experiments on 7DoF robotic arm.



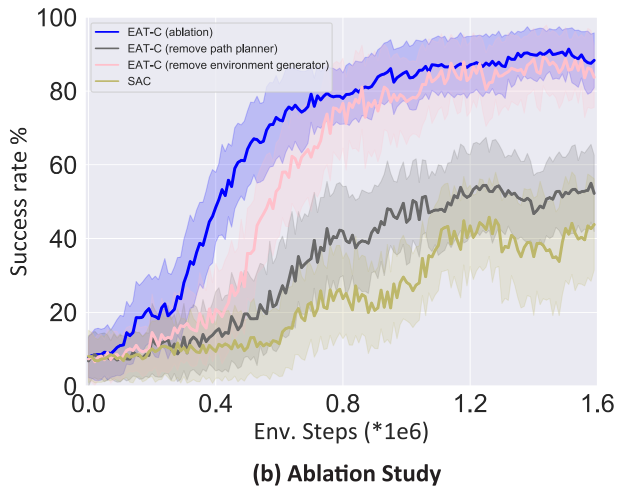


Figure 7: **(a)** report the success rate (mean±std averaged over 6 random seeds) of EAT-C and baselines on test tasks in 2D Pusher environments. **(b)** Ablation study of EAT-C on 2D Pusher tasks.

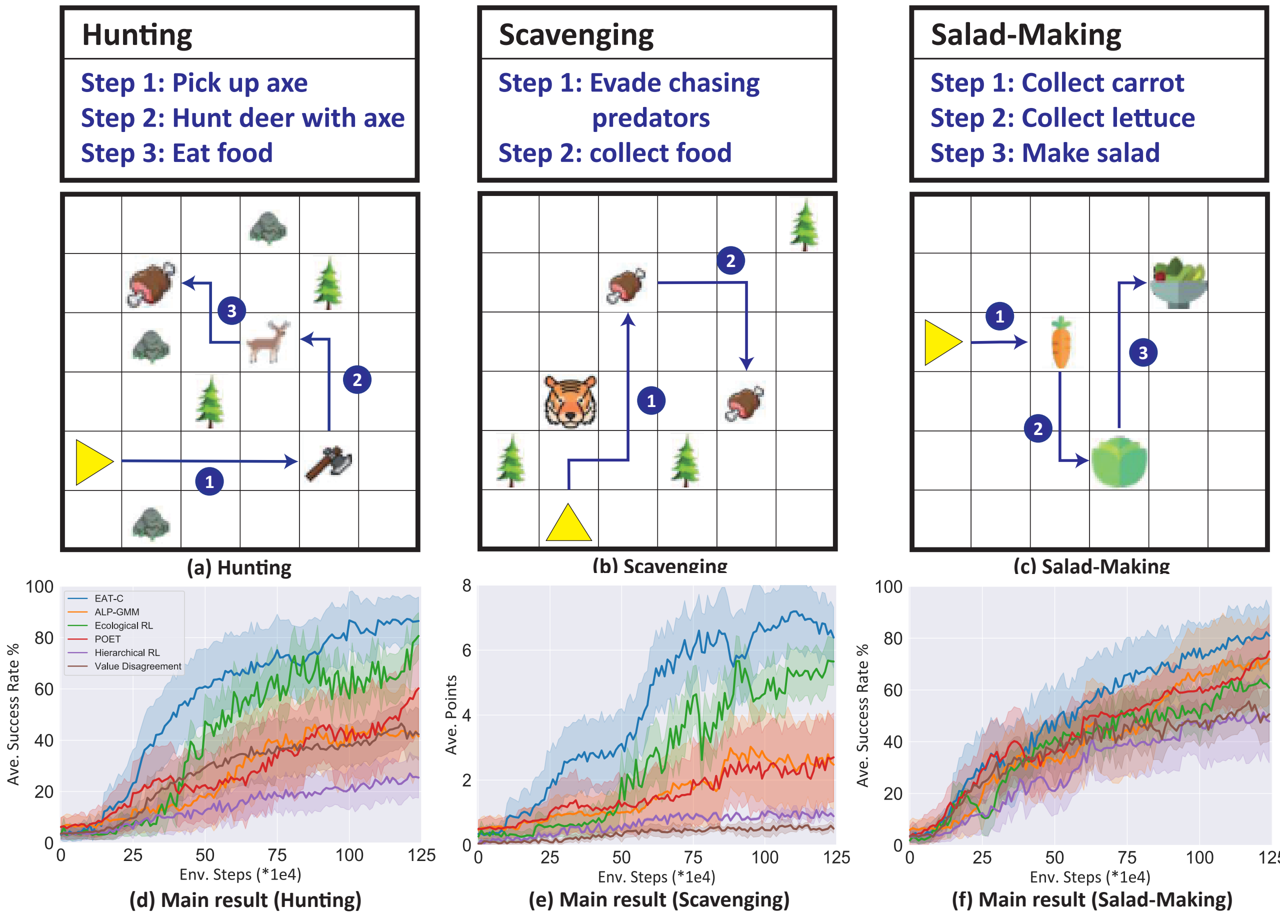


Figure 8: **(a)-(c)** illustrate the 2-3 key steps for completing each task. In Scavenging, the agent will have 2 points when it collects food each time. **(d)-(f)** report different methods' performance (mean±std over 10 random seeds) on multiple test tasks.

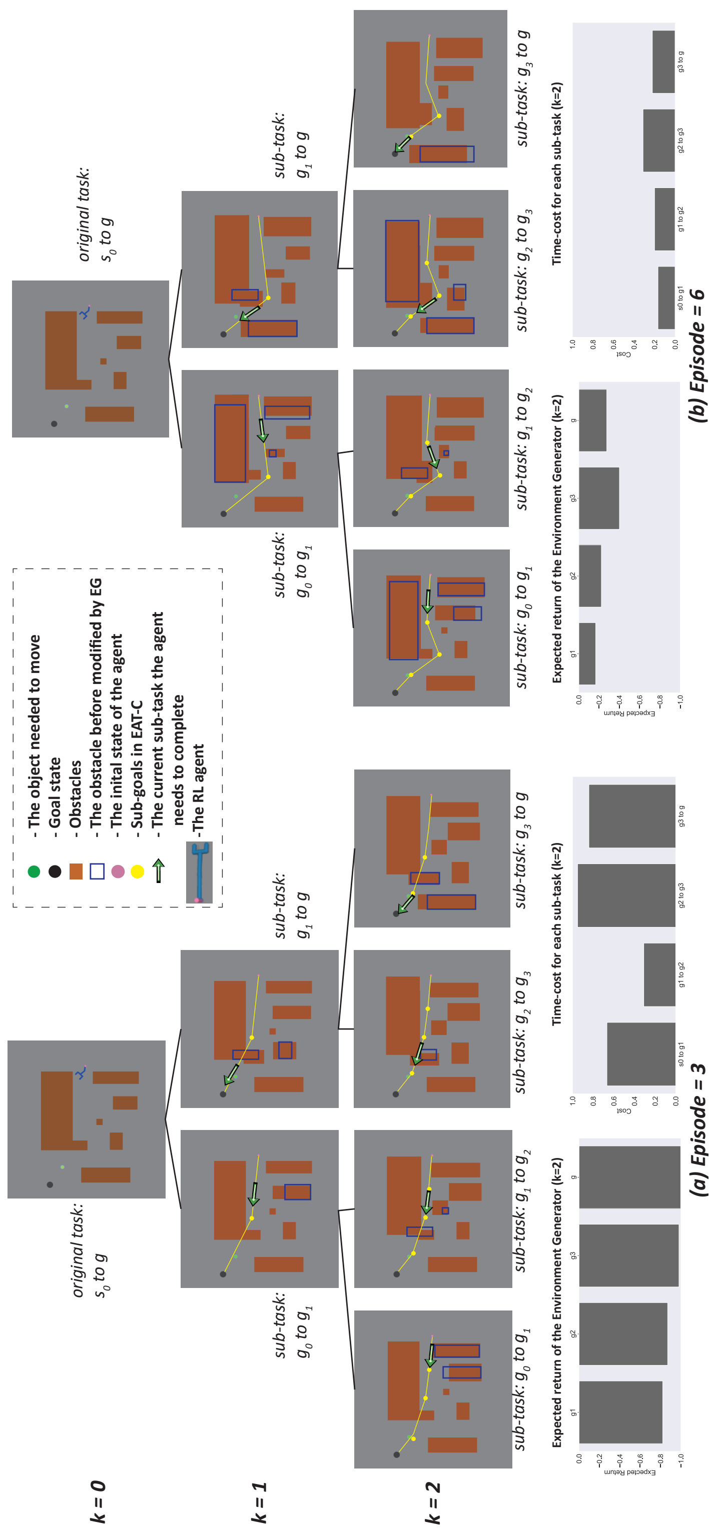


Figure 9: Visualization of EAT-C. A 2D robot with a 4-joint arm starts from the initial state (pink), navigates to the object (green) location, and then pushes the object to the goal state (black). The histograms in (a) and (b) represents the expected return of EG taking action b_t, and the costs of sub-tasks predicted by the path planner in layer $k = 2$, respectively.

D.3. Additional Analysis on Main Results and Ablation Studies

Analysis on main results: Refer to the results reported in Fig 7, Fig 8, and Table 7, by comparing the methods with environment changed, we notice that controlling the difficulty of the modified environments is critical to earlier-stage learning efficiency. In particular, both EAT-C and POET trains another agent (e.g., the path-planner in EAT-C and the antagonist agent in POET) to control the hardness of the training tasks matching the capability of the RL agent, so their earlier-stage improvement than others. In contrast, the environments selected in ALP-GMM might be too challenging and the ones modified by POET might be too easy, leading to poorer efficiency in earlier stages. Although these methods are designed to train RL policies that can adapt to different environments or tasks, only EAT-C trains a path-planner to decompose a long-horizon task into a curriculum of easy-to-hard sub-task sequences for training. The guidance of path-planner and its curriculum plays a critical role in outperforming these baselines.

Analysis on ablation study: To evaluate the generalization and robustness, which are the advantages of EAT-C due to the adversarial environment generator (EG), we evaluate different methods on **multiple new random environments during the test phase**. This is different from the ablation study in Fig. 3(b), which evaluates all methods on **the fixed training environment**. The new results show a large gap (80.24 vs. 42.23) between EAT-C and EAT-C (remove EG). This demonstrates that EG is important to improving the generalization and robustness of the RL policy. In the training environment (non-random) used in our original ablation study of Fig. 3(b), Hierarchical RL (HRL) does improve the performance of the default RL algorithm (SAC) on long-horizon tasks, i.e., 39.62 (SAC) vs. 68.27 (HRL). However, in random and unseen environments, HRL generalizes much poorer than EAT-C.

Test Setting	Multiple New Random Environments	Training Environment
EAT-C	80.24 ± 12.25	92.04 ± 6.49
EAT-C (remove EG)	42.23 ± 10.34	85.47 ± 9.12
EAT-C (remove Planner)	27.58 ± 14.67	46.02 ± 10.3
SAC	20.83 ± 7.24	39.62 ± 12.25
Hierarchical RL	22.04 ± 10.44	68.27 ± 6.99

Table 8: Ablation Study Results (larger version of Table. 2)

D.4. Ablation Study: Whether EG will make reward more sparse?

In EAT-C, EG can improve the learning efficiency of the RL agent by adversarially modifying the environment. This may raise two essential questions: **(1)** whether EG could make the environment more reward sparse? **(2)** whether planner could always generate infeasible sub-goals? To answer these questions, in Fig. 10, we report the average time-cost that the agent needs to complete each sub-task in layer-3 of the sub-task tree.

- In earlier stages when π_p and the RL agent are not well trained, π_p may generate hard sub-tasks. However, after a little training on the sub-task curriculum, π_p is trained to generate a minimum-cost path for the RL agent and the capability of the RL agent to finish the sub-tasks is also improved.
- We apply EG to simple sub-tasks that are optimized to be simple in EAT-C (via optimizing the planner) for the RL agent. The goal of EG is to avoid learning similar and easy sub-tasks repeatedly, which cannot provide informative feedback to RL even if the reward is dense. On the other hand, we set several restrictions to avoid over-adversarial environments, as mentioned in Appendix. A

D.5. Ablation Study: What if EG generate unfeasible environments?

Considering that modifying/generating environments will result in unfeasible ones, we conduct connectivity check on the modified/generated environments for EAT-C (Line.7, Algorithm. 5) and baseline methods. The connectivity test is common in navigation as well as many complicated and realistic environments/tasks (Francis et al., 2020; Yingjun & Xin-wen, 2019). For example, in maze tasks like 2D-pusher, modifying the obstacles easily results in unsolvable environments. Whether to apply the test is a property of an environment/task instead of a limitation of our method only: to avoid wasting computation on unsolvable ones, most methods adopt the connectivity test by default when applied to such an environment/task. For some simpler environments/tasks, e.g., BipedalWalker mainly used in the original ALP-GMM and POET paper, connectivity

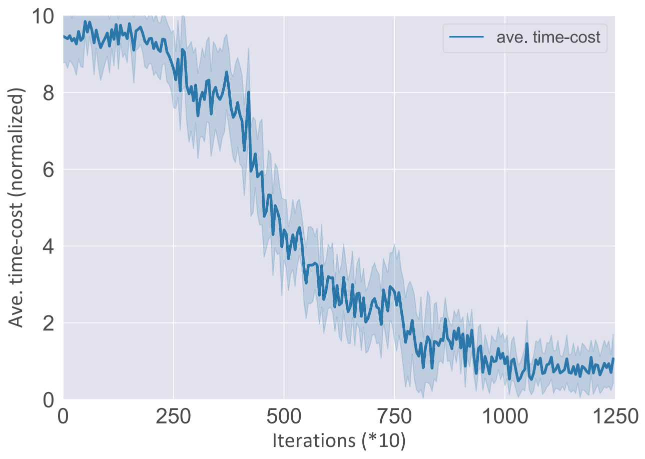


Figure 10: Average time-cost of the RL agent to complete a sub-task from layer-3 of the sub-task tree. As the training proceed, time-cost that the agent needs to complete each sub-task decreases significantly, indicating that π_p does not propose infeasible goals, and EG does not make the reward more sparse.

test is not required by any method. However, as we show in the table 9, ALP-GMM and POET generate more unsolvable environments than EAT-C when applied to 2D-pusher.

Env. Steps	2.0 million	8.0 million	10.0 million
EAT-C	6.046 %	2.015 %	1.131 %
POET	10.889 %	2.510 %	1.586 %
ALP-GMM	13.740 %	3.490 %	1.697 %

Table 9: Results of EAT-C and curriculum RL methods generating unfeasible environments during training. Due to the low fail possibility of generating unfeasible environments, removing connectivity check will not heavily change the final training results.

Removing the connectivity test will not heavily change the final training results because the agent gets no effective reward from unsolvable environments. However, removing the test does affect the efficiency because the agent has to waste time on unsolvable environments. As the new experiments we will show later, EAT-C generates fewer unsolvable environments than other baselines, so removing the test will not change the advantage of EAT-C on training efficiency.

Due to the nature of the environments in this paper, we applied the connectivity test to every method evaluated, so the comparison is fair to all methods. To evaluate how these methods' efficiency is affected by the removal of the connectivity test, in the table below, we report the percentage of unsolvable environments generated/sampled by different methods at different stages of the training. It shows that (1) the unsolvable environments generated by all the three methods drastically decrease to $< 2\%$ after 10 millions steps so they only affect the efficiency of early training stages; (2) EAT-C generates much less unsolvable environments than other baselines so it is still more efficient when the connectivity test is removed.

D.6. Ablation Study: What if combine hierarchical RL with other curriculum RL method?

One main difference between EAT-C and other curriculum RL that can also modify the environments is: **EG in EAT-C locally modify environments rather than change the entire environment once.** Training an environment generator (EG) to locally modify environments for a curriculum of sub-task is easier and results in more efficient learning because:

- EG does not need to create a curriculum or fully explore the whole environment space, which is highly expensive and challenging in other environment design methods;
- Generating does NOT add complexity compared to other environment design methods. On the opposite, it REDUCES

the complexity because easier sub-tasks can provide denser and more informative feedback than the original long-horizon task.

We conduct an empirical experiment to illustrate the advantages of the designing of EG in EAT-C and report in Table. 10. In this ablation experiments, we replace EG in EAT-C with POET so both the path-planner and the RL agent are trained within the environments generated/mutated by POET. This is "EAT-C (EG $\rightarrow$ POET)" in table 10. Note that "EAT-C (EG $\rightarrow$ POET) maintains a hierarchical structure of sub-tasks generated by the path-planner. Thus, it can also be regarded as an example of combining hierarchical RL (HRL) with curriculum RL (POET).

Env. steps	$\mathbf{2.5}$ million steps	$\mathbf{5.0}$ million steps	$\mathbf{7.5}$ million steps	$\mathbf{10.0}$ million steps
EAT-C (original)	22.07 ± 10.55	39.76 ± 13.46	61.24 ± 16.13	69.45 ± 17.42
EAT-C (EG $\rightarrow$ POET)	16.44 ± 8.21	37.18 ± 14.25	49.94 ± 12.21	57.64 ± 10.93
POET	20.32 ± 11.35	39.22 ± 10.79	43.10 ± 14.54	50.23 ± 12.83

Table 10: In this experiment, we compare the performance of training RL policy in diverse environments mutated by POET (EAT-C (EG $\rightarrow$POET)) with the original EAT-C and POET. Note that EAT-C (EG $\rightarrow$ POET) has a hierarchical sub-task structure generated by the path-planner. From the result, we prove that locally modifying the environment (EAT-C) allows the RL agent learns more efficiently than change the entire environment once.

The results show that applying some curriculum generated by existing methods such as POET to HRL (i.e., EAT-C (EG $\rightarrow$ POET)) can finally outperform POET but it is less efficient than POET in early stages (before 5.0 million steps), because of the expensive and inefficient exploration of the environment space discussed above. On the other hand, our original EAT-C significantly outperforms both POET and EAT-C (EG $\rightarrow$ POET) on learning efficiency and final performance. Hence, our sub-task curriculum with adversarial environments is more efficient than some existing curricula applied to HRL.

D.7. Ablation Study: Curriculum RL controls everything rather than environment only

An insight of curriculum learning method is that the generated curriculum should be able to control everything during training (i.e., training tasks, initial state, and training environments), which is possible to lead to a better performance than control training environments only. Therefore, we conducted an ablation experiments of curriculum RL (e.g., ALP-GMM) that can control everything (initial and goal states, obstacles, object) in 2D-pusher. Specifically, ALP-GMM can control the location and size of the obstacles, the initial/goal state, and the location of the object by sampling from the learned GMM. We report the performance of the new **"ALP-GMM (control all)"** on test tasks over the course of training in the table 11. We also include our previous results of EAT-C and ALP-GMM for comparison. Note these two cannot control the original tasks, i.e., the initial state s_0 and the final goal state g.

This ablation experiments show that **(1)** EAT-C with partial control still significantly outperforms ALP-GMM (control all), i.e., 81.45 vs. 61.05; and **(2)** ALP-GMM (control all) can surpass ALP-GMM with partial control in the middle stage of training (54.52 vs. 48.15 at 10.0 million env. steps) but its final performance is worse (61.05 vs. 66.21). This can be explained by our analysis in the response to your first comment: curriculum having total control over the assigned tasks can introduce bias and distribution drift over the course of online RL. Therefore, randomly drawing the assigned tasks but building a curriculum within each task, which is how EAT-C works in our evaluation setting, is more robust.

Environment Steps	$\mathbf{5.0}$ million steps	$\mathbf{10.0}$ million steps	$\mathbf{15.0}$ million steps	20.0 million steps
ALP-GMM	18.33 ± 14.25	48.15 ± 11.34	62.35 ± 8.47	66.21 ± 9.35
ALP-GMM (control all)	34.28 ± 12.14	54.52 ± 14.33	58.67 ± 10.54	61.05 ± 12.45
EAT-C	39.76 ± 13.46	69.45 ± 17.42	78.15 ± 13.66	81.45 ± 11.35

Table 11: Ablation Study Results in 2D pusher. In this ablation experiment, ALP-GMM (control all) can control the location and size of the obstacles, the initial/goal state, and the object's location by sampling from the learned GMM. Results show that even though the generated curriculum can control more, the performance is not better.

E. More Related Work

The sub-goal generation in (Pertsch et al., 2020) follows a top-down and coarse-to-fine manner. However, they need to search for each sub-goal in the tree from many possible candidates, **which is expensive and requires a search tree** (hierarchical partition of the whole sub-goal space) much larger than our sub-goal tree (see Eq. 2). On the contrary, EAT-C learns a planning policy to **directly generate su-bgoals** and we do not need to build the search tree covering the whole sub-goal space. Another major difference is that **they study a planning-only method while we study a mutual learning strategy** between planning and RL to improve both planning and RL policies.

(Zhang et al., 2020) trains a high-level policy to find the shortest path of sub-goals in a trained adjacency space. However, the distance between any two points in the adjacency space is expected to reflect the time cost of the agent navigating between the two points in the environment, which **can be very challenging or even infeasible to achieve in many tasks** (If we have such an adjacency space, both planning and RL can have dense feedback and simple supervised learning should work). In contrast, EAT-C trains a planner to directly generate a min-cost path of sub-goals through an easy-to-hard curriculum (fewer sub-goals interpolated at first), which provides an easier and more efficient solution **without requiring learning an adjacency space**. Moreover, the data used to train the planner in EAT-C are more informative than (Zhang et al., 2020) and cover multi-granularity since they are collected from RL when completing the bottom-up sub-task curriculum.

In (Dayan & Hinton, 1993), the high-level managers set a sequence of subgoals in the environment partitioned by Euclidean distance, which does not consider the obstacles or the RL agent capability. Hence, **there is no mutual training between high-level (planner) and low-level (controller) managers in (Dayan & Hinton, 1993)**. On the contrary, the planner in EAT-C is jointly trained with the RL agent to produce a min-cost path of sub-goals for RL, which results in a more efficient curriculum of sub-tasks to train the RL agent.

(Zhang et al., 2020) and (Schmidhuber & Wahnsiedler, 1993) plan sub-goals sequentially from the starting state to the goal state, which might be inefficient in complex tasks (requiring expensive search in a large space) and **cannot produce the easy-to-hard curricula on a sub-goal tree as in EAT-C**. In contrast, we train a planner to recursively produce coarse-to-fine sub-goal trajectories between the starting and goal states, which naturally provide an easy-to-hard curriculum for every component.