UNIVERSITY OF TECHNOLOGY SYDNEY
Faculty of Engineering and Information Technology

Auto-Generated Curriculum For Reinforcement Learning

by

Shuang Ao

A Thesis Submitted
in Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

Sydney, Australia

2023

Certificate of Original Authorship

I, Shuang Ao, declare that this thesis is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the School of Computer Science, Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

SIGNATURE: Production Note: Signature removed prior to publication.

DATE: 15th Aug, 2023

PLACE: Sydney, Australia

ABSTRACT

Auto-Generated Curriculum For Reinforcement Learning

by

Shuang Ao

This research investigates the efficacy of a curriculum constructed to train reinforcement learning for the purpose of improving learning efficiency, generalization, and robustness in challenging tasks. we focus on an automatically generated curriculum that is designed to train a Reinforcement Learning (RL) agent through a sequence of easy-to-hard sub-tasks and thus leads to the development of an RL policy that exhibits superior learning efficiency in tasks of the long horizon and sparse rewards. Additionally, our research demonstrates an RL policy of greater generalization and robustness in diverse environments and agents with different morphologies.

Over the past few decades, reinforcement learning (RL) has demonstrated its efficacy in addressing control and decision-making challenges across a range of domains, including but not limited to robotics, video games, and autonomous vehicles. However, targeting long-horizon tasks can be highly inefficient due to the sparsity of rewards. The issue of sparse rewards leads to insufficient feedback from the agent's exploration, thereby diminishing the efficacy of the learning process. One of the main obstacles in the field of RL pertains to improving its generalization and robustness to environmental variations. Specifically, a policy that has been trained in a particular environment may exhibit susceptibility to even minor alterations in the deployed environment, leading to suboptimal generalization in practical applications. Thus, we develop an auto-generated curriculum as a potential solution to address the primary challenges encountered in RL. This approach aims to improve the learning efficiency of training an RL policy and promote the generalization of the trained policy to diverse environments. More specifically, the auto-generated cur-

riculum improves the RL policy in the following directions: First, a cross-training framework is presented that utilizes planning and RL to create a sequence of subtask training curricula that gradually increase in difficulty, thereby improving the learning efficiency of RL. Then, the proposed framework incorporates adversarial modifications to the training environment, which improves the adaptability of the trained RL policy to diverse environments.

Subsequently, the present study conducts comprehensive experiments on standard tasks to corroborate the efficacy of the proposed curriculum from both a quantitative and qualitative standpoint. The benchmark tasks, including maze navigation, robotic control, and cross-environment validation, have the ability to comprehensively assess the models' capacity to apply acquired knowledge to novel tasks and environments. The resulting empirical findings demonstrate that the proposed models have attained state-of-the-art performance across a diverse set of tasks, while also exhibiting superior learning efficiency.

Despite the generalization of RL to diverse tasks and environments, the current community is deficient in research pertaining to the control, design, or evolution of agent morphology in RL. The practical implementation of RL in engineering and research holds immense importance. The agent's morphology frequently draws upon bionics, however, the development of precise structural and dynamic parameters is a costly and labor-intensive process. Conversely, the matter of optimizing the robot's structure to suit various surroundings remains an unresolved issue. This study delves into the integration of robot morphology optimization into the training process of RL policy, building upon the aforementioned curriculum generated automatically. Our proposal outlines a mechanism for co-evolution between environment and morphology. This involves modifying the training environment during the evolution of the agent's morphology, allowing the morphology to adapt to varying environments. Experimental verification was conducted in various scenarios, and the findings demonstrated that our approach can facilitate swift morphological evolution while also guaranteeing the state-of-the-art performance of the RL policy across diverse environments. Furthermore, a comprehensive ablation study is conducted

to validate the rationality of the morphology-environment co-evolution framework.

Key Words: Reinforcement Learning, Curriculum Learning, Sub-task Generation, Morphology Evolution

Dissertation directed by A/Prof. Jing Jiang, A/Prof. Guodong Long, and Dist/Prof. Chengqi Zhang
Australian Artificial Intelligence Institute
Faculty of Engineering and IT
University of Technology Sydney

Acknowledgements

First of all, many thanks to my principal supervisor Prof.Jing Jiang, and co-supervisor Dist. Prof. Chengqi Zhang and Prof. Guodong Long. And many thanks also to my co-worker, Tianyi Zhou, from the University of Maryland. During chasing the degree of Ph.D., they gave me careful guidance and kind care in my study, research, and life. Their profound knowledge, rigorous academic attitude, and tireless study spirit made me admire, and left an indelible impression on me, which greatly influenced and inspired me. I will bear their edification in my mind, and these will definitely benefit me for my whole life. I also greatly appreciate the research center, Australian AI Institute at the University of Technology Sydney provides a great platform to support my research works.

I also want to thank my colleagues and group mates, Haiyan Zhao, Zhuowei Wang, Zhihong Deng, Tao Shen, Lu Liu, ..., for their help in all aspects of model implementation and thesis writing. They gave me tremendous help with the questions I raised and put forward many valuable suggestions for my codes and thesis, which benefit me a lot.

Lastly, I am deeply grateful to my parents, Xian Ao, and Ping Wang. Every the progress I have made is condensed by their selfless dedication, support, and encouragement. Special thanks to my fiancée, Ziye Zhou. I cannot become who I am today without you. Thanks to every teacher who has taught me the growth of patient teaching; thanks to all my friends who mutually helped to grow up together and the friendship deserves my cherish. These lovely people will continue to encourage me to move forward for more solid and comprehensive work in the future.

Shuang Ao

Sydney, Australia, 2023.

List of Publications

Published Conference Paper:

C-1 Shuang Ao, Tianyi Zhou, Guodong Long, Qiuhua Lu, Lumin Zhu, Jing Jiang, CO-PILOT: COllaborative Planning and reInforcement Learning On sub-Task curriculum, NeurIPS 2021. (Core A*)

C-2 Shuang Ao, Tianyi Zhou, Jing Jiang, Guodong Long, Xuan Song, Chengqi Zhang, EAT-C: Environment-Adversarial sub-Task Curriculum for RL. ICML 2022. (Core A*)

C-3 Shuang Ao, Tianyi Zhou, Guodong Long, Xuan Song, Jing Jiang, Curriculum Reinforcement Learning via Morphology-Environment Co-Evolution. Submitted to NeurIPS 2023.

Contents

List of Figures

Abbreviation

Reinforcement Learning - RL

Artificial Intelligence - AI

Curriculum Learning - CL

Generative Adversarial Network - GAN

Recurrent Neural Network - RNN

Markov Decision Process - MDP

Partially Observed Markov Decision Process - POMDP

Deep Neural Network - DNN

Convolutional Neural Network - CNN

Gated Recurrent Unit - GRU

Long Short-Term Memory - LSTM

Maximum Likelihood Estimation - MLE

Chapter 1

Introduction

1.1 Background

Although AI can surpass humans on certain tasks, humans still perform much better in making sequential decisions via learning from interactions with the environment. Reinforcement learning (RL)) (Sutton and Barto, 2018) aims to bridge this gap by learning to optimize the trajectories of agents (e.g., controllers, robots, game players, self-driving cars, etc) to achieve the maximal return. However, in complicated long-horizon tasks, RL usually suffers from poor sample efficiency and costly data collection. Moreover, the data quality is often low due to sparse rewards when rollouts fail and cannot provide informative feedback. Model-based RL and off-policy RL improve the sample complexity with the price of extra biases, causing unstable and brittle optimization. Instead of reaching a single goal, goal-conditioned RL (Kaelbling, 1993) learns one model for any given goal input to its model(s). However, it needs to be trained to reach many possible goals, and the resulting model's performance still degrades drastically for distant goals.

On the other hand, 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. A promising strategy to overcome this problem is to train the agent on multiple tasks in different environments (Wang et al., 2019a; Gur et al., 2021) via multi-task learning or meta-learning (Salimans et al., 2017; Finn et al., 2017). However, it increases

the training cost and the space for possible environments/tasks can be too large to be fully explored by RL. So how selecting the most informative and representative environments/tasks to train an RL agent to evolve generalizable skills becomes a critical open challenge.

Unlike RL agents that do not actively seek new environments to improve their learning capability, natural species have full motivations to do so to survive in the competitive world, and one underlying mechanism to drive them is evolution. Evolution is a race with the changing environment for every species, and a primary goal is to accelerate its adaptation to new environments. Besides merely optimizing its control policy, evolution more notably changes the morphology of species, i.e., the skeletal structure and the attributes for each part, in order to make them adapt to the environment. In fact, the improvement on morphology can be more critical because there could exist a variety of actions or skills an agent cannot do (no matter how the control policy is optimized) without certain structures, e.g., more than one leg, a long-enough limb, or a 360-degree rotation joint.

1.1.1 Curriculum for Reinforcement Learning

Curricula have a ubiquitous role in the early stages of human development, formal educational systems, and continuous learning that extends into adulthood. The acquisition of skills in various domains, such as sports or mathematics, involves a systematic and planned training procedure. This approach is designed to introduce new concepts and tasks in a sequential manner, building upon the foundation of previously acquired knowledge. The significance of curriculum quality in achieving success has been proven across various domains of human learning. Attempting to impart knowledge of integral or derivative concepts to a 3-year-old lacking fundamental arithmetic skills appears to be an exceedingly challenging endeavor. Education plays a pivotal role since it offers a systematic approach to deconstructing

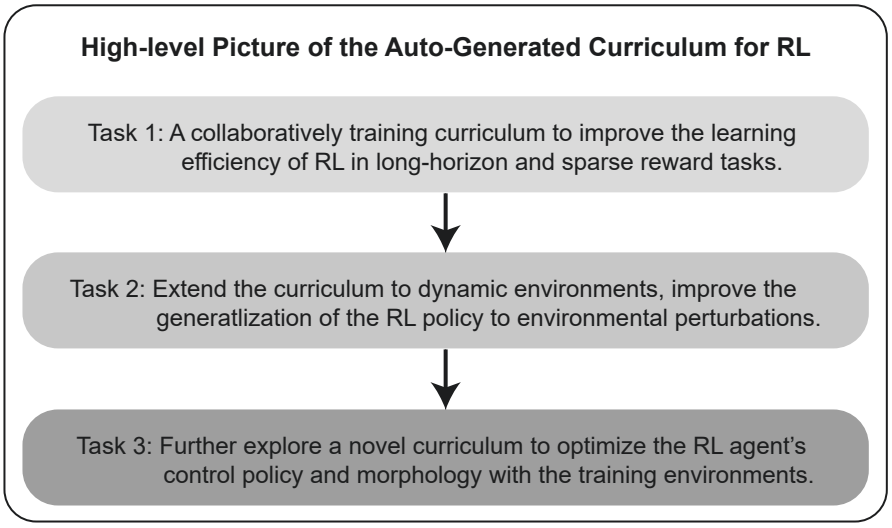

Figure 1.1 : **High-level picture of the tasks in this thesis.** This thesis addresses three challenges encountered in the training of RL policy, i.e., low learning efficiency due to long-horizon and sparse reward, sensitivity to environments, and the agent's ability to generalize across diverse environments.

intricate knowledge and a structured curriculum that facilitates the gradual acquisition of concepts, progressing from simpler to more challenging levels. A curriculum facilitates the acquisition of knowledge and understanding by people, particularly in relation to complex subjects, by becoming them more approachable and comprehensible. By employing a curriculum, it is possible to enhance the efficiency of training machine learning models. In the year 1993, Elman (1993) introduced the concept of curriculum-based training for neural networks. The initial research conducted by the individual focused on the acquisition of basic language grammar, highlighting the importance of a specific approach. This approach involved commencing with a restricted collection of uncomplicated data and gradually introducing more intricate training samples. Failure to adhere to this strategy resulted in the model's inability to acquire the desired knowledge. In contrast to training without a structured curriculum, it is expected that the incorporation of a curriculum will expedite the

process of convergence and improve the overall performance of the final model.

Our research is motivated by the RL community's prevalence of the aforementioned challenges, i.e., sparse reward, highly sensitive to environments and poor generalization. However, as stated previously, it is extremely difficult to solve these problems using RL alone. Consequently, we combine curriculum learning to enhance the efficiency of learning and the final effect of RL in relevant scenarios. In stark contrast to previous approaches to curriculum learning, the training curriculum in our framework is entirely generated and adapted to the reinforcement learning training process. This concept's primary justification is depicted in Fig. 1.1. The present study seeks to address the challenges of long-horizon and sparse rewards, which hinder the effectiveness of RL, by implementing a collaborative training curriculum. On the basis of this foundational knowledge, we then expand the scope of the training to improve the adaptability of RL in various environments. Finally, additional research is required to determine the relationship between agent morphology and the training environment. As a proposed remedy, we suggest implementing a co-evolutionary program that takes morphology and environment into account. The objective of this curriculum is to improve the morphology of the RL agent in order to facilitate its adaptability to a spectrum-spanning variety of environments. In the following three sections, we will delve deeper into these scenarios, challenges in tasks, and framework design of corresponding methods.

1.1.2 Sparse Reward in Long-horizon Tasks

Reinforcement learning (RL) faces a common challenge known as the sparse reward problem. In RL, an agent learns to make decisions and take actions in an environment to maximize a cumulative reward signal. However, in certain scenarios, the reward signal provided to the agent is sparse, meaning that it is infrequent or only given upon reaching a specific goal state.

The sparse reward problem arises in situations where the desired behavior or the optimal policy requires the agent to navigate through a complex environment with only occasional feedback. This poses a significant challenge for RL algorithms, as they struggle to explore and learn in an efficient manner when the reward signal is sparse.

One of the primary issues with sparse rewards is that they provide limited guidance to the agent during the learning process. Without frequent feedback, the agent may have difficulty discerning which actions or states are desirable and which are not. Consequently, the learning process becomes slower and less effective, as the agent must explore a vast state space without clear indications of progress or success.

The sparse reward problem is often encountered in real-world domains where the rewards are delayed or occur only upon reaching specific goals. For example, in a robotics task, the reward might be given only when the robot completes a task successfully or reaches a desired target state. Similarly, in games, rewards may be sparse, with positive feedback given only upon achieving certain milestones or winning conditions.

Addressing the sparse reward problem is a crucial area of research in RL. Various techniques have been developed to tackle this challenge and enable agents to learn effectively even in the presence of sparse rewards. These techniques include reward shaping, where additional auxiliary rewards are designed to guide the agent's behavior toward the desired goals, and curiosity-driven exploration, where agents are incentivized to explore novel or uncertain states to discover new information.

Another approach to addressing sparse rewards is the use of hierarchical RL (Kulkarni et al., 2016), which involves decomposing complex tasks into subgoals with more frequent rewards. This allows the agent to learn incrementally, with each subgoal

providing a more manageable learning signal. Furthermore, recent advancements in deep RL, such as the use of intrinsic motivation or unsupervised pre-training (Pathak et al., 2017), have shown promise in dealing with the sparse reward problem. These approaches aim to encourage agents to explore and learn based on internal curiosity or by leveraging pre-existing knowledge in the absence of explicit rewards.

Planning algorithms are usually more robust and effective on long-horizon tasks. Given a distance metric, they discretize the state space to a grid/graph and seek for the shortest collision-free path between states using graph search such as Dijkstra's algorithm or A* (Hart et al., 1968). Thereby, it only needs a local policy to navigate between consecutive states on the path. However, it is challenging to learn or estimate the distance accurately in complicated tasks such as mazes. Moreover, planning every step on the path is as difficult as the original RL and requires fine-grained discretization impractical for high-dimensional states. Planning only a few milestone states leaves the RL agent to solve relatively long-horizon subtasks. Although sampling-based search heuristics can build a graph with a better exploration-exploitation trade-off, they are not optimized for the RL policy. Eysenbach et al. (2019a) adapt planning to a learned RL policy, which can provide distances estimated from its replay buffer, but the performance largely depends on the RL policy and its exploration.

To address the aforementioned problem, we first introduce "CO-PILOT", a collaborative learning scheme between planning and goal-conditioned RL. As illustrated in Figure 1.2, it trains each model under the other's guidance along a curriculum of sub-tasks. Unlike most existing planning methods, we train a planning policy to recursively decomposes a task into two easier sub-tasks, which finally yields a tree containing coarse-to-fine trajectories of sub-goals to the final goal. The tree naturally forms a curriculum for more effective training. During the top-down tree construction, we start by training the planner to find the shortest path on a coarser

graph with fewer sub-goals, which is an easier training task, and gradually request it to generate detailed paths with denser sub-goals. We measure the distance by the cost of an RL agent navigating between consecutive sub-goals, so the planner is optimized to produce the most efficient path for the RL agent.

1.1.3 The Generalization of RL Policy to Environments

A significant hurdle in the field of reinforcement learning (RL) is the development of agents that can effectively generalize their acquired policies to diverse environments. The concept of generalization pertains to the capacity of a reinforcement learning agent to effectively apply its acquired knowledge and competencies in novel and unobserved settings. This ability enables the agent to maintain high-performance levels despite encountering unfamiliar environments.

The ability to generalize reinforcement learning policies across diverse environments is of paramount importance due to the prevalence of variations, uncertainties, and changes in real-world scenarios. Merely training a reinforcement learning (RL) agent to perform optimally in a particular environment may lead to restricted adaptability and a decline in performance when confronted with diverse environments.

In order to attain generalization, reinforcement learning agents must acquire knowledge representations, strategies, or policies that effectively encapsulate the fundamental patterns and dynamics of the environment, rather than relying solely on memorization or specific details of training instances. In order to effectively perform in various environments, agents are required to identify and utilize the fundamental structure of the problem at hand.

There exist multiple factors that contribute to the challenge of generalization in the field of Reinforcement Learning (RL). The challenge of navigating high-dimensional state spaces presents a significant obstacle for reinforcement learning agents due to the curse of dimensionality. Efficient generalization from a restricted

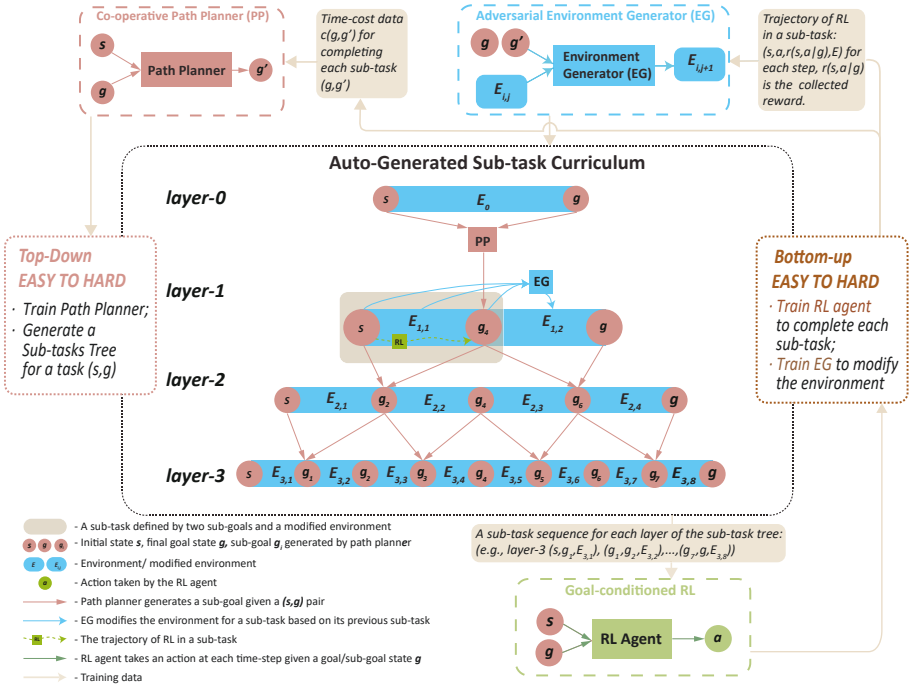

Figure 1.2 : **Main Structure of CO-PILOT and EAT-C:** In CO-PILOT, the planner is trained to recursively decompose a task (s, g) to a sub-task tree of coarse-to-fine min-cost sequences of sub-tasks. While this top-down construction forms an easy-to-hard curriculum to train the planner, a bottom-up traversal of those sub-tasks forms an easy-to-hard curriculum for RL. The planned sub-tasks provide dense rewards enabling more efficient RL, while RL's cost on each sub-task is used to train the planner for producing more cost-efficient sub-tasks for RL. In EAT-C, we further include an environment generator (EG) to adversarially modify 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.

set of training samples is a crucial requirement. Moreover, the existence of stochasticity or noise within the environment can exacerbate the process of generalization. It is imperative for agents to acquire the ability to make decisions in a resilient manner, even in the face of unforeseeable fluctuations.

The objective of reinforcement learning (RL) is to instruct an agent to acquire an optimal policy that exhibits high performance in a designated environment. Notwithstanding the effectiveness and proficiency of a well-trained policy in a given environment, a prevalent issue arises when it encounters minor changes or variations, leading to a failure in generalization and robust performance. The phenomenon is referred to is commonly recognized as the sensitivity of a proficiently trained reinforcement learning policy to variations in the environment.

The issue of sensitivity arises as a result of the intrinsic intricacy and fluctuation of environments in the real world. Minor alterations, such as adjustments in the dynamics, transition probabilities, or initial conditions, can exert a substantial influence on the efficacy of the agent. Thus, a policy that is tailored for a particular environment may not be able to effectively adjust or extend to accommodate even minor changes in the environment. The challenge of sensitivity poses a significant obstacle for reinforcement learning algorithms, as it limits the potential for learned policies to be applied and scaled in real-world situations. The capacity of reinforcement learning (RL) systems to effectively manage variations, uncertainties, or novel situations that are commonly encountered in dynamic environments is constrained. There exist multiple factors that contribute to the issue of sensitivity. RL agents frequently depend on particular assumptions or models of the environment throughout the learning phase. Departures from these assumptions may result in behavior that is suboptimal or erroneous. The challenge posed by the curse of dimensionality is a significant concern in the field of reinforcement learning. This is due to the fact that RL agents are required to explore and learn in state spaces that are often

quite large, which can make it difficult to generalize from limited training samples. Insufficient exploration during training may limit the agent's exposure to a variety of environmental conditions, thereby impeding its capacity to adapt to alterations.

To address this challenge and improve policy's robustness and generalization to small changes in the environments. A growing number of studies (Pinto et al., 2017; Vinitsky et al., 2020; Ferguson and 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., 2019b), it can make long-horizon tasks even more challenging and reward-sparse, hence detrimental to RL's efficiency.

Then, we extend "CO-PILOT" to be more adaptive to diverse environments. Thus, 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. 1.2, The new 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.

1.1.4 Morphology Optimization to Diverse Environments

Although current RL can excel on a specified task in a fixed environment through massing training, it usually struggles to generalize to unseen tasks and/or adapt to new environments. A promising strategy to overcome this problem is to train the agent on multiple tasks in different environments (Wang et al., 2019c; Portelas et al., 2019a; Gur et al., 2021; Jaderberg et al., 2017) via multi-task learning or meta-learning (Salimans et al., 2017; Finn et al., 2017). On the other hand, transfer learning techniques enable RL agents to leverage knowledge and policies learned in one task or environment to improve their performance in new, unseen tasks or environments. By pre-training on related tasks or domains, the agent can transfer its learned representations, policies, or value functions to expedite learning and adaptation in different scenarios. Meta-learning, on the other hand, trains agents to acquire meta-knowledge or meta-policies that enable rapid learning and adaptation to new environments. Both transfer learning and meta-learning provide mechanisms to enhance generalization and adaptability in RL.

Domain generalization (Liang et al., 2020) aims to train RL agents on multiple source domains with different characteristics, allowing them to learn policies that are robust to variations in those domains. By exposing agents to diverse environments during training, they develop the ability to generalize their learned policies across different but related environments. This approach helps RL agents perform well in unseen environments by leveraging the commonalities and shared knowledge across

domains.

Reward shaping (Laud, 2004) and curriculum design (Hafner et al., 2018; Wulfmeier et al., 2015) can also help in this problem. By carefully designing the reward structure and curriculum in RL, agents can be guided to learn policies that generalize better to unseen tasks or environments. Reward shaping involves adding auxiliary or shaping rewards that provide additional guidance to the agent during learning. These shaping rewards can help highlight important aspects of the task and encourage the exploration of beneficial behaviors. Curriculum design, as mentioned earlier, involves progressively increasing the difficulty or diversity of the training tasks, enabling agents to gradually generalize to more challenging environments.

However, it increases the training cost and the space for possible environments/tasks can be too large to be fully explored by RL. So how selecting the most informative and representative environments/tasks to train an RL agent to evolve generalizable skills becomes a critical open challenge. For RL, we claim that **A good morphology should improve the agent's adaptiveness and versatility, i.e., learning faster and making more progress in different environments.** From prehistoric persons to modern Homo sapiens, there is a definite association between the rise of civilization and the Homo sapiens' optimization of their physical form for improved tool use. Unfortunately, the morphology in many RL researches is predefined and fixed so it could be sub-optimal for the targeted environments/tasks and may restrain the potential of RL. Although morphology can be optimized using RL (Sims, 1994; Wang et al., 2019d; Kurin et al., 2021; Yuan et al., 2022) to improve the final performance for a specific task or environment, it was not optimized for the adaptiveness to varying environments. Moreover, instead of re-initializing the control/RL policy after every morphology modification or environment change, the millions of years of evolution demonstrate the effectiveness of continual and progressive learning of the control policy over generations of morphology evolution along

with a sequence of varying environments. How to optimize the morphology and control policy of an RL agent for the above goal is still an underexplored problem.

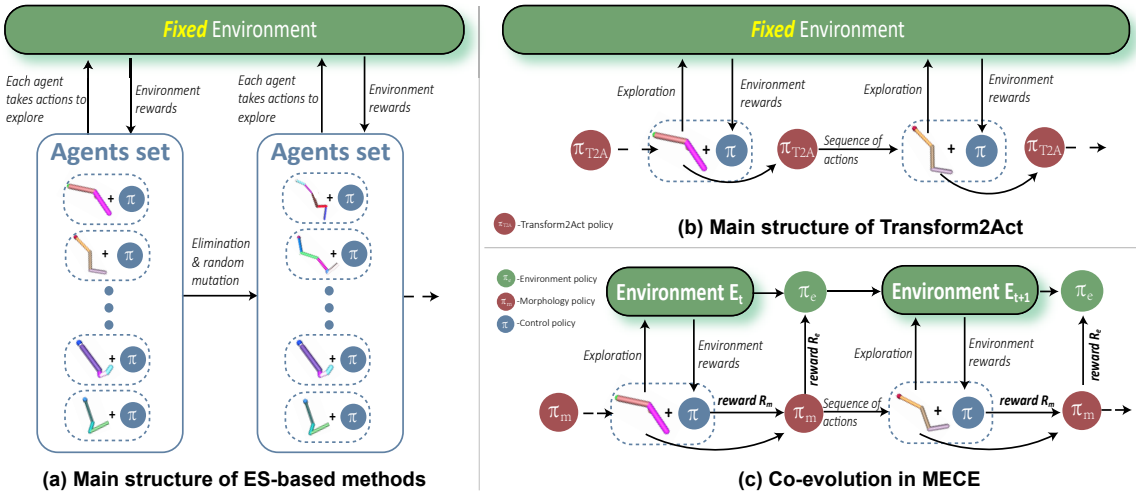

Figure 1.3 : **MECE vs. previous methods.** (a) Evolution strategy (ES)-based method for morphology optimization. It starts with agents with different morphology, eliminates those with poor performance, and then applies random mutation to the survivors. (b) Transform2Act, which trains an RL policy to modify the morphology of an agent in a fixed environment. (c) MECE (ours). We train two policies to optimize the morphology and change the environment to form a curriculum over the course of training a control policy.

Given that the agent has the freedom to improve its morphology and control policy for faster adaptation to different environments/tasks, the remaining question is: what environment can improve this learning process and incentivize the agent to keep finding better morphology (rather than staying with the same one)? The learning environment plays an essential role in RL since it determines the data and feedback the agent can collect for its training. Thus, we adopt a simple criterion: **A good environment should accelerate the evolution of the agent and help it find better morphology sooner,** which will enhance the agent's adaptiveness and versatility over new environments/tasks.

The interplay between morphology and environment has lasted for billions of years. In the endless game, species that find the appropriate environments to evolve their morphology and policy may survive and keep being improved via adaptation to new environments/tasks. Inspired by this, we propose an *morphology-environment co-evolution (MECE)* scheme that automatically generates a curriculum of varying environments to optimize the agent's morphology and its RL policy so they can be generalized to unseen environments and adapted to new morphology, respectively. Inspired by the former discussion of "good morphology" and "good environment", we optimize the morphology policy to improve the agent's adaptiveness+versatility and optimize the environment policy to improve the morphology's evolution. To this end, we design two rewards for the two policies based on the learning progress of the control policy, which is estimated by comparing its concurrent performance on a validation set and its historical trajectories. Therefore, the three policies are complementary and mutually benefit each other: (1) the morphology and environment policies create a curriculum to improve the generalizability of the control policy; (2) the environment policy creates a curriculum to optimize the morphology policy; (3) the control policy provides rewards to train the other two policies. In Fig. 1.3, we illustrate the co-evolution process of MECE and compare it with evolution-strategy (ES) based method (Wang et al., 2019d) and Transform2Act (Yuan et al., 2022), which uses RL to optimize the morphology in a fixed environment. In MECE, we train the control policy and automatically determine when to apply the other two policies to modify the morphology or change the environment: they are triggered by the slow progress on the current morphology and environment, respectively.

1.2 Research Objectives

The primary aim of this thesis is to develop an auto-generated curriculum for reinforcement learning (RL) with the goal of improving its learning efficiency and

overall performance in complex tasks. In order to accomplish this goal, we have identified three composite objectives that specifically address the aforementioned challenges:

i. To address the challenges associated with reinforcement learning training in a sparse reward environment, including low learning efficiency and low sampling efficiency.

ii. To address the issue of a well-trained policy being susceptible to minor environmental perturbations.

iii. To address the problem of the deployment of a trained RL policy to various agents and environments.

1.3 Contributions of This Thesis

This thesis primarily focuses on the training and learning challenges associated with RL. These challenges include but are not limited to low sample efficiency, inadequate exploration in complex environments, and low learning efficiency in tasks with sparse rewards. Our primary focus lies in investigating methods to enhance the efficacy of RL in intricate real-world environments and tasks. Additionally, we want to enhance the generalization capabilities of reinforcement learning across various tasks and environments. This thesis demonstrates the efficacy of integrating curriculum learning as a means to enhance training efficiency and improve performance in complicated environments and tasks. In this thesis, we propose a set of curriculum learning algorithms that possess self-generating capabilities. These algorithms are designed to optimize task sequences adaptive to the training phase of RL, rather than pre-defined by humans. In the experimental section, a multitude of extensive experiments were conducted to substantiate the efficacy of the method we devised. Simultaneously, we conduct thorough ablation studies to compare and demonstrate

the rationale behind the design of each component in the algorithm. We present visualizations of the findings for each algorithm, aiming to enhance readers' comprehension of how the automatically created curriculum can successfully support RL during the training phase.

1.4 Content and Organisation

This thesis is organized as follows:

- Chapter 2: This chapter will present a survey of reinforcement learning and related work of three concepts that have been involved in the research.
- Chapter 3: This chapter presents a novel curriculum framework of a collaborative training scheme for an RL agent and a path planner.
- Chapter 4: This chapter extends the former curriculum framework to be able to adapt to diverse environments.
- Chapter 5: This chapter presents a curriculum framework to train generalizable RL, whose morphology and policy are optimized for different environments.
- Chapter 6: This chapter summarizes the thesis contents, elaborates its contribution, and also provides several future research directions of applying curriculum learning for RL.

Chapter 2

Literature Review

The following literature review provides a brief introduction to three topics that are closely related to my research.

2.1 Reinforcement Learning and Planning algorithm

Goal-conditioned RL Pong et al. (2018); Kaelbling (1993); Schaul et al. (2015a) took a goal as an additional input to its model(s) and aims to handle different goals/tasks using the same policy. However, it requires more exploration and expensive training on various possible goals, and it still easily fails to reach distant goals in practice. Goal relabeling and reward shaping (Andrychowicz et al., 2017a; Nasiriany et al., 2019) have been studied to mitigate these issues by modifying the task rewards to be dense but they introduce extra data 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) trained a goal/task generator based on the agent's performance on existing tasks. But it is usually non-trivial to determine and tune the appropriate difficulty level for each training stage. Recent methods (Florensa et al., 2019; Wu et al., 2019) improved it by learning a compact representation of the goal space. The goal-conditioned value function $V(s|g)$ naturally provides an ideal distance metric for shortest path planning. Dennis et al. (2020); Wang et al. (2019a) proposed to train RL policy on a curriculum of environments adaptive to the RL. In CO-PILOT, we train a sub-goal tree planner to generate a sample-efficient and adaptive training curriculum that trains goal-conditioned to reach distant goals

progressively. On the other hand, the goal-conditioned policy's cost on the planned paths is used to improve the planning policy.

Planning Sutton and Barto (2018); LaValle (2006) were more effective in addressing long-horizon tasks in practice (Levine et al., 2011a; Kavraki et al., 1996). It usually refers to dynamic programming that finds the optimal path between two nodes on a graph. Planning methods in RL, e.g., value/policy iteration (Lau and Kuffner, 2005a; Levine et al., 2011a), utilized or learned an environment model to improve the RL policy. Compared to a reactive policy, an advantage of planning is that the planned trajectory provides a global view of future steps. However, learning the environment model usually requires expensive exploration of state space (Elbanhawi and Simic, 2014; Lavalle, 1998; Hsu et al., 1997; Pertsch et al., 2020), structured and compact modeling of the environment/graph (Chen et al., 2020a; Rickert et al., 2008), and an accurate distance metric (Eysenbach et al., 2019a). Sequentially planning (Schmidhuber and Wahnsiedler, 1993; Zhang et al., 2020) sub-goals from the starting state to the goal state 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., 2011b; Lau and Kuffner, 2005b; Elbanhawi and Simic, 2014), which can be as challenging as model-free RL. Control-based methods require accurate models for both the robot and the environment (Howard and Kelly, 2007; Werling et al., 2012), as well as an accurate distance metric (Eysenbach et al., 2019b), which can be a rather daunting task.

Hierarchical planning (HP) Nasiriany et al. (2019); Jurgenson et al. (2020a); Pertsch et al. (2020) searched for a sequence of sub-goal on a tree to guide the exploration of an agent but building the hierarchical partition of all possible sub-goals can be expensive. A conventional hierarchical planning algorithm (Kaelbling and Lozano-Pérez, 2011) also learns to predict sub-tasks of a tree structure based on a set of pre-defined motion primitives. Hierarchical RL (HRL) for goal-reaching

tasks has been recently studied in (Zhang et al., 2021; Nachum et al., 2018). Shu et al. (2018) trained the RL agent on a human-designed curriculum of tree-structured sub-goals. They learn a sequence of primeval policies towards finishing complicated tasks, where the higher-level policies decompose a complex task into easier sub-tasks or motion primitives that can be addressed by the lower-level policies or controllers. However, the hierarchy of sub-tasks need to be either determined by prior knowledge or discovered automatically, which is usually challenging due to the huge space of possible sub-tasks. In contrast, EAT-C trains one planning policy to decompose a hard task into sub-tasks of multiple difficulty levels by only using the RL's time cost data. It directly generates the sub-tasks and thus requires neither hierarchical partition of the task space nor a predefined set of sub-tasks.

2.2 Combining Planning with RL

A line of recent works (Amos et al., 2018; Lee et al., 2018a; Srinivas et al., 2018; Schrittwieser et al., 2019) embeded a planning model as one part of an RL agent's model and train it together with the RL policy in an end-to-end manner. Chiang et al. (2019); Faust et al. (2018); Savinov et al. (2018) found that combining the two can help agents to reach distant goals in specific tasks. Eysenbach et al. (2019a); Savinov et al. (2018) proposed planning strategies with graph search based on the replay buffer of experiences from a given RL policy. Schrittwieser et al. (2019) proposed to use Monte-Carlo tree search when planning in latent space to achieve a better optimization on value function. These results inspire our work, but our primary difference is the mutual training between RL and planning, which does not require either a pre-trained policy or strong heuristics about the distance metric. In CO-PILOT, both are trained from scratch and can mutually boost and guide each other's training via an auto-generated curriculum of easy-to-hard sub-tasks. This mutual training leads to a principled learning framework adaptive to a vast amount

of potential applications. Several recent works Wang et al. (2019b); Portelas et al. (2019b); Gur et al. (2021) showed that the efficiency and generalization of RL can be improved by generating/selecting different environments for training. Wang et al. (2019b) studied a method "POET" to pair an adversarial environment generator with two agents to improve the generalization of the RL policy to novel environments. Portelas et al. (2019b) proposed a teacher algorithm to sample environments based on a Gaussian mixture model capturing both the diversity of environments and the learning progress on them. This leads to a curriculum improving the efficiency of RL.

2.3 Curriculum Learning

Vanilla Curriculum Learning (CL). The concept of Vanilla CL was first introduced in Bengio et al. (2009a). The authors demonstrated that machine learning models exhibit enhanced performance when exposed to progressively challenging training samples. In this paper, Vanilla CL, also referred to as CL, is defined as a method of sample selection that relies solely on the curriculum as a rule-based criterion. Commonly, the field of Continual Learning utilizes a predetermined set of regulations to differentiate between straightforward and challenging instances.

Self-paced learning (SPL) is distinguished from conventional curriculum learning (CL) by the manner in which examples are assessed. To be more precise, the primary distinction concerning Vanilla Continual Learning pertains to the sequence in which the samples are presented to the model. The order in SPL is not predetermined, but rather determined based on the model's performance, and as such, it is subject to change throughout the training process. The level of complexity is assessed iteratively throughout the training phase, with modifications made to the sequence of samples. Kumar et al. (2010) employs the probability of the prediction to establish the ranking of the samples, whereas Lee and Grauman (2011) utilized

objectness to determine the order of training.

Self-paced curriculum learning (SPCL) refers to a learning approach that allows individuals to progress through a curriculum at their own pace. As previously mentioned in the introduction of this section, it is evident that a significant degree of overlap between categories is not only plausible but also a common occurrence. The categories of SPL and CL merit explicit acknowledgment, as numerous works are utilizing both in tandem. The present study aims to highlight the SPCL approach, which involves the combined utilization of predetermined standards and metrics based on learning to determine the sequence of sample training. The aforementioned paradigm was initially introduced by Jiang et al. (2021) and subsequently utilized in the domains of matrix factorization and multimedia event detection. Furthermore, it has been utilized in various other tasks, including but not limited to weakly-supervised object segmentation in videos, as demonstrated by Zhang et al. (2017).

Progressive CL. The concept of Progressive CL (PCL) pertains to a method of curriculum design that does not rely on the difficulty level of individual samples. Rather, it involves a gradual modification of the model capacity or task settings to facilitate progressive learning. The PCL approach does not adhere to the implementation of CL in terms of sample order. Rather, it employs the curriculum concept to a related task or a particular segment of the network. This results in a simpler task at the outset of the training process, which gradually increases in difficulty towards the end. Morerio et al. (2017) presented an instance of the Curriculum Dropout technique, wherein a monotonic function is formulated to reduce the likelihood of dropout occurrences during the training process. The aforementioned approach is aimed at enhancing the overall performance of the model. According to the authors, the initial stage of the training process may exhibit a low dropout rate, which is expected to gradually increase in order to achieve a substantial enhancement in performance levels. An additional illustration for this classification is the method-

ology put forth in the work of Karras et al. (2017), which involves incrementally augmenting the capability of Generative Adversarial Networks (GANs) to attain superior outcomes.

Teacher-student CL. The pedagogical approach known as teacher-student collaborative learning (TSCL) involves dividing the instructional process into two distinct components. Specifically, this method entails the use of a primary model, referred to as the "student," which is responsible for acquiring knowledge and skills related to the primary learning objective. Additionally, a secondary model, known as the "teacher," is employed to determine the optimal learning parameters for the student. The curriculum is executed through a network architecture that employs a policy on a student model, culminating in the ultimate inference. The proposed methodology was initially introduced by Kim and Choi (2018) utilizing deep reinforcement learning techniques. Subsequently, this approach was redefined in subsequent studies (Hacohen and Weinshall, 2019; Jiang et al., 2017; Zhang et al., 2019).

2.4 Curriculum Learning for RL

Task Generation. The problem of generating intermediate tasks specifically for a curriculum is known as task generation. Narvekar et al. (2016) introduced a number of techniques for generating intermediate tasks in preparation for a particular final task. Methods are dependent on a definition of a domain as a collection of MDPs identified by a task descriptor, which is a vector of parameters specifying the degrees of freedom in the domain. Based on Object-Oriented MDPs, da Silva and Costa (2018) proposed an analogous procedure for partially automated task generation in their curriculum learning framework. There is an assumption that each task has a class environment that is parameterized by a number of attributes. A function, which must be provided by the designer, generates simpler versions of the ultimate task by removing unnecessary steps. instantiating the attributes with

values that make the tasks easier to solve.

Schmidhuber (2011) proposed the Powerplay framework, which aims to cultivate a more versatile problem-solving agent through the generation of novel issues in an unsupervised manner. The system endeavors to identify a novel task as well as an alteration to the existing problem-solving mechanism, such that the updated solver is capable of successfully addressing all previously encountered tasks in addition to the newly introduced one. The search algorithm (Srivastava et al., 2012) operated on a problem-specific encoding of the domain and the solver, and its effectiveness has been validated by its application in tasks related to pattern recognition and control. The generator and solution of the problem are provided with a restricted computing budget, hence prioritizing the creation of the most basic tasks that were previously unsolvable.

Task sequencing. The process of arranging tasks in a specific order to achieve a desired outcome is commonly referred to as task sequencing. The objective of sequencing is to arrange a given set of tasks or samples in a manner that optimizes the learning process. A multitude of sequencing techniques are available, each with distinct underlying presumptions. One of the underlying premises of curriculum learning posits that it is possible to manipulate the learning environment in order to generate diverse tasks. The extent of control wielded by a practitioner in implementing curriculum learning is a determining factor in the suitability of various sequencing techniques.

Schaul et al. (2015b) introduced the concept of Prioritized Experience Replay (PER), a technique that prioritizes and replays significant transitions with greater frequency. Transitions that exhibit a high anticipated learning progress are deemed significant, as determined by their temporal difference (TD) error metric. Replaying samples with larger temporal difference (TD) errors can potentially facilitate

more substantial updates in the network, as per intuition. As individuals acquire knowledge of transitions, there is a shift in the distribution of significant transitions, resulting in an implicit educational program across the various examples.

Kim and Choi (2018) proposed an additional modification to prioritized experience replay, which involves the incorporation of a second neural network to jointly learn the weight or priority of a sample with the main network. The auxiliary network, denoted as ScreenerNet, is trained to forecast weights based on the discrepancy between the sample's output and the main network's output. In contrast to PER, the present approach is characterized by its memoryless nature, enabling it to make direct predictions regarding the importance of a given training sample, irrespective of whether or not that specific example was previously encountered. Hence, the aforementioned methodology has the potential to solicit experience tuples in an active manner, which could yield maximum information or utility, thereby facilitating the development of an online curriculum. An alternative approach to using sample importance as a sequencing metric is to reorganize the training process according to the encountered sample trajectories. In the process of learning, it is common to encounter easily attainable states initially, while more challenging states are encountered at a later stage in the learning process.

In practical scenarios characterized by sparse rewards, it is possible that the easily accessible states may not yield a reward signal. The utilization of the Hindsight Experience Replay (HER) (Andrychowicz et al., 2017b) is a viable strategy to optimize the benefits of these initial encounters. The HER approach is characterized by its ability to assimilate knowledge from both the intended and unintended outcomes. This is accomplished by reenacting each episode with a focus on the actual goal accomplished, rather than the one initially pursued by the agent.

Co-learning. Co-learning is a pedagogical strategy that involves multiple agents

or versions of the same agent interacting within a shared environment to generate a curriculum. The agents have the ability to act in a cooperative or adversarial manner, with the ultimate goal of acquiring new behaviors. This process results in the development of an implicit curriculum, which facilitates the improvement of both sets of agents over time. The paradigm in question encompasses various methodologies, one of which is self-play. This methodology has yielded several noteworthy outcomes, including TD-Gammon (Tesauro, 1995), as well as more recent examples such as AlphaGo (Lillicrap et al., 2016) and AlphaStar (Vinyals et al., 2019).

Rather than offering a comprehensive overview of all literature pertaining to the utilization of self-play or co-learning, we have opted to showcase a limited number of scholarly articles that focus on the formulation of objectives for multiple agents in order to facilitate co-learning. Sukhbaatar et al. (2017) introduced a novel approach known as asymmetric self-play, which enables an agent to acquire knowledge about its surroundings in an unsupervised fashion, without the need for external rewards. This approach involves the involvement of two entities, namely an educator and a learner, operating under the framework of "the instructor presenting an assignment, and the pupil executing it." Simultaneous learning of policies by two agents is achieved through the maximization of interdependent reward functions for tasks that are based on goals. The responsibility of the educator is to maneuver towards a state of the environment that the learner will utilize either as an objective, in case the environment can be reset, or as an initial state if the environment can be reversed.

Pinto et al. (2017) proposed the Robust Adversarial RL (RARL) approach, which involves training a pair of agents in a single-agent RL task using adversarial techniques for policy learning. In contrast to the approach of asymmetric self-play (Sukhbaatar et al., 2017), which involves the teacher setting the objective for the student, the RARL method involves training a protagonist and an adversary.

The protagonist is trained to accomplish the original RL task while also being able to withstand the disruptive forces introduced by the adversarial agent.

Initial/Terminal state changes. We examine approaches that explicitly create different MDPs for intermediate tasks, by changing some aspect of the MDP. One of the earliest examples of this type of method was learning from easy missions. Asada et al. (2005) proposed this method to train a robot to shoot a ball into a goal based on vision inputs.

Florensa et al. have recently introduced more comprehensive techniques for conducting the aforementioned reverse expansion. The authors put forth the concept of reverse curriculum generation, which involves the utilization of an algorithm to produce a distribution of initial states that progressively diverge from the desired outcome. The methodology presupposes the existence of at least one known goal state, which serves as a starting point for the process of expansion. Proximate initial states are produced through the implementation of a stochastic process, whereby a random walk is taken from pre-existing initial states, with actions being selected in the presence of noise perturbations. To determine the subsequent set of initial states for expansion, an estimation of the anticipated return is conducted for each of these states. The selection process involves identifying those states that yield a return within a predefined range of minimum and maximum intervals. This particular interval has been calibrated to facilitate the expansion of states in which progress can be made while avoiding the pitfall of making progress too effortless. Ivanovic et al. (2018) employed a comparable methodology that integrates the reverse expansion phase for curriculum development with physics-based priors to hasten the learning process of continuous control agents.

Florensa et al. (2019) also explored an approach for expansion in the opposite direction of "forward". This approach enables an agent to autonomously identify

diverse objectives within the state space, thus facilitating the exploration of said space. The present study employed a Generative Adversarial Network (GAN) as proposed by Held et al. (2018). Specifically, the generator network is utilized to suggest goal regions that are parameterized subsets of the state space, while the discriminator is responsible for assessing the suitability of the goal region in terms of its level of difficulty relative to the agent's current ability.

The specification of goal regions is accomplished through the utilization of an indicator reward function, while policies are conditioned on both the state and the goal, akin to a universal value function approximator as described by Schaul et al. (2015b). The agent undergoes training on tasks that are recommended by the generator.

2.5 Curriculum of training in different environments

Several recent works (Wang et al., 2019b; Portelas et al., 2019b; Gur et al., 2021) showed that the efficiency and generalization of RL can be improved by training the policy in different environments. They (Portelas et al., 2019b; Wang et al., 2019b) trained 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. Rajeswaran et al. (2017) presented a method that combines deep RL and human demonstrations to train an agent for complex manipulation tasks. By leveraging both RL and expert demonstrations, the agent learns to perform dexterous manipulation in diverse environments, achieving higher success rates and improved generalization. Florensa et al. introduced a reverse curriculum learning approach where RL agents start with complex tasks and grad-

ually move to simpler ones. By focusing on more challenging environments early in training, the agents acquire robust policies that generalize well to a wide range of diverse environments.

A sample-efficient RL algorithm (Lee et al., 2018b) that combined episodic learning and backward updates. The algorithm enables RL agents to learn policies more efficiently by utilizing experiences from multiple episodes and transferring knowledge to diverse environments. Liu et al. (2019) introduced a multi-agent RL approach where agents learn to cooperate or compete to achieve their objectives. By training agents in competitive environments. Lowrey et al. (2018) focused on transferring RL policies learned in simulation to physical robotic systems for non-prehensile manipulation tasks. The authors propose an approach that enables successful transfer of policies, allowing RL agents to perform diverse manipulation tasks in the real world.

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.

2.6 Morphology Design and Control

Continuous Design Optimization. A line of research in the community has studied optimizing an agent's continuous design parameters without modifying its skeletal structure, and they commonly optimize on a specific kind of robots. Baykal and Alterovitz (2017) studied an annealing-based framework for cylindrical robot optimization. Ha et al. (2017); Desai et al. (2017) optimized the design of legged robots by trajectory optimization and implicit function. Applying deep RL into design optimization becomes more popular recently. Chen et al. (2020b) used com-

putational graphs to model robot hardware as part of the policy. CMA-ES Luck et al. (2019) optimized robot design via a learned value function. Ha et al. (2017) proposes a population-based policy gradient method. Another line of work Yu et al. (2019); Exarchos et al. (2021); Jiang et al. (2021) searched for the optimal robot parameters to fit an incoming domain by RL. Different from the prior works, our approach learns to find the most general skeleton of an agent while maintaining its continuous design parameters optimization to diverse environments or tasks.

Combinatorial morphology Optimization. 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. 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) trained 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 and 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 and 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 and Wahnsiedler (1993) planned 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.

One closely related line of work is the design of modular robot morphology spaces and developing algorithms for co-optimizing morphology and control (Sims, 1994; Liao et al., 2019; Gupta et al., 2022; Trabucco et al., 2022) within a design space to find task-optimized combinations of controller and robot morphology. When the control complexity is low, evolutionary strategies have been successfully applied to find diverse morphologies in expressive soft robot design space (Cheney et al., 2013, 2018). For more expressive design spaces, GNNs have been leveraged to share controller parameters (Wang et al., 2019d) across generations or develop novel heuristic search methods for efficient exploration of the design space (Zhao et al., 2020). In contrast to task specific morphology optimization, Hejna et al. (2021) proposed evolving morphologies without any task or reward specification. Hejna et al. (2021)

employ an information-theoretic objective to evolve task-agnostic agent designs.

Chapter 3

Collaborative Training Curriculum of Planning and Reinforcement Learning

3.1 Preliminaries and Basic settings

3.1.1 Goal-conditioned Reinforcement Learning

Goal-conditioned RL or multi-goal RL learns a policy that can be adapted 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 uses $\pi(a|s,g)$ to interact with an environment described 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.

In each episode, the agent starts from an initial state $s_0 \sim p_0(s)$ and aims to reach a given goal $g \in \mathcal{G}$. In every time step t, it takes an action $a_t = \pi(a|s_t, g)$ (deterministic) or $a_t \sim \pi(a|s_t, g)$ (stochastic), receives a reward $r(s_t, a_t|g)$, and moves to a new state $s_{t+1} \sim p(s'|s_t, a_t)$. RL aims to learn a policy π maximizing the expected return $\mathbb{E}_{(s_0,g)}[\mathbb{E}_\pi(R_0)]$. 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 $V(s|g) \triangleq \mathbb{E}(R_t|s_t = s, g) = \mathbb{E}_{a\sim\pi}[Q(s,a|g)] = \sum_{a\in\mathcal{A}} \pi(a|s,g)Q(s,a|g)$. Directly maximizing the expected return or V w.r.t. π results in the vanilla policy gradient method (Sutton and Barto, 2018), which usually samples inefficient and suffers from the high variance of R_t. Actor-critic meth-

ods (Sutton et al., 1999) additionally learns a model of V or Q as a "critic" to the "actor" π, which performs as a baseline to effectively reduce the variance. The optimization of V or Q aims to minimize the Bellman residual

$$J_{Q^\pi} = \mathbb{E}_{(s_t,a_t,g)} \left[Q^\pi(s_t, a_t|g) - r(s_t, a_t|g) - \gamma \mathbb{E}_{s_{t+1}}[V(s_{t+1}|g)] \right]^2, \tag{3.1}$$

Given the critic Q, maximizing the expected return w.r.t. π reduces to minimizing

$$J_\pi = \mathbb{E}_{(s_t,g)}[-V(s|g)] = \mathbb{E}_{(s_t,g)}[\mathbb{E}_{a_t}[-Q^\pi(s, a|g)]]. \tag{3.2}$$

A typical actor-critic algorithm alternates between minimizing J_Q and J_π. To encourage exploration, we use soft actor-critic (SAC) (Haarnoja et al., 2018a) that augments V with an entropy term (with temperature α), i.e.,

$$V(s|g) = \mathbb{E}_{a\sim\pi}[Q^\pi(s, a|g) - \alpha \log \pi(a|s, g)]. \tag{3.3}$$

In order to encourage the above equation, SAC additionally optimizes V by minimizing the mean square error

$$J_V = \mathbb{E}_{(s_t,g)} \left[V(s_t|g) - \mathbb{E}_{a\sim\pi}[Q^\pi(s_t, a_t|g) - \alpha \log \pi(a_t|s_t, g)] \right]^2. \tag{3.4}$$

SAC alternatively optimizes J_V, J_Q and J_π (using the augmented V in Eq. (3.3)) defined in Eq. (3.1)-(3.4) by using stochastic gradient decent (SGD) on batches of sampled (s_t, a_t, g). Although we use SAC in our experiments, any off-policy RL algorithm can replace it in our framework.

3.1.2 Reward Shaping For RL by Path-Planning

In various environments, an RL agent receives a nonzero reward only when reaching the ϵ-ball $B(g, \epsilon)$ around the goal g, i.e., $r(s, a|g) = \mathbb{1}[s \in B(g, \epsilon)]$ with $\mathbb{1}$ being the indicator, so $r(s, a|g)$ for most steps/trajectories cannot provide informative feedback to policy training. RL is unstable and can easily fail with such sparse reward, especially in long-horizon tasks when g is far away from s_0

or too difficult to reach for the agent-in-training. To address this problem, reward shaping method (Laud, 2004) augments the environment reward with a dense reward $r'(s, a|g)$ that can be issued to more non-goal states, e.g., intrinsic motivation/curiosity that encourages effective exploration, or human-engineered task-specific rewards. The ideal dense reward, which is, however unavailable without knowing π^*, is $V^*(s|g)$. Planning methods, e.g., value iteration (Tamar et al., 2016) or fitted-Q iteration (Antos et al., 2008), can approximate $V^*(s|g)$ but accurately estimating $V^*(s|g)$ is as challenging as the policy learning.

Path-planning and motion-planning methods (Elbanhawi and Simic, 2014) usually adopt a heuristic distance or cost $c(s, g)$ (e.g., Euclidean distance or time cost) to replace the unknown $V^*(s|g)$. They discretize the state space into a grid/graph and find the shortest path connecting the initial state s_0 and the goal g. In goal-conditioned RL, s_0 and g can be any feasible states on the graph, so path-planning needs to solve the all-pairs shortest path(APSP) problem (Russell and Norvig, 2003), i.e.,

$$\min_{g_0=s_0, g_{1:T-1}, g_T=g} \sum_{t=0}^{T-1} c(g_t, g_{t+1}), \quad \forall s_0 \in \mathcal{S}, g \in \mathcal{G}, \tag{3.5}$$

where $g_{1:T-1}$ denotes a discrete sequence of sub-goals $(g_1, g_2, \ldots, g_{T-1})$ between $g_0 = s_0$ and $g_T = g$. Planning every step for an agent is usually challenging and requires an accurate $c(\cdot, \cdot)$ or environment model. But we only use planning for reward shaping, which can tolerate a small T, i.e.,

$$\bar{r}(s, a|g_{0:T}) = \frac{1}{T} \sum_{t=1}^{T} r(s, a|g_t). \tag{3.6}$$

As T increases, $\bar{r}(s, a|g_{0:T})$ becomes denser and the RL agent can receive more effective feedback for policy training. However, the quality of $\bar{r}(s, a|g_{0:T})$ also heavily depends on the cost for the agent to reach g by following the path $g_{0:T}$, since issuing reward to an inefficient/long path misleads the policy training. Therefore, in COPILOT, we use a prediction model to predict $c(\cdot, \cdot)$ and train the planning policy to

generate $g_{0:T}$ incurring the smallest cost as in Eq. (3.5).

Remarks: This leads to a collaborative learning and mutual boosting scheme between RL and planning: planning produces easy sub-tasks that the RL agent can complete in a few steps and thus provide dense rewards enabling more efficient RL, while the time costs of RL agent on those sub-tasks can be used to further improve the planning policy towards producing more cost-efficient paths and better reward shaping. In addition, this scheme makes both RL and planning easier to overcome their bottlenecks: RL **l**earns from dense rewards to complete long-horizon tasks, while relatively coarse (with small T) planning suffices to provide dense rewards so every-step planning relying on accurate modeling of MDP is not necessary. We will introduce more details next.

3.2 Collaboratively Training Framework for RL agent and Planner

In our training scheme, planning serves RL like a copilot in an aircraft to encourage more efficient training. The main advantage of a tree structure planner is to provide a global view of future milestones to the RL policy, which mainly focuses on local steps and might lack long-term sight. However, many planning algorithms are based on Bellman equation and sequentially predict the sub-goals, which may suffer from accumulated errors (Ross et al., 2011). In addition, as the aforementioned, a larger T results in easier sub-tasks for the RL agent but also increases the difficulty of planning, and vice versa. Hence, it is challenging to train both the RL and planning policy from scratch using either a small or a large T. This motivates us to seek coarse-to-fine planning that can generate multiple trajectories of sub-goals with increasing T, so the planning policy can be trained on an easy-to-hard curriculum (Bengio et al., 2009b; Fang et al., 2019), i.e., generating coarse-to-fine shortest paths from small T to large T. At the same time, the RL agent can also be trained

on an easy-to-hard curriculum of sub-tasks, i.e., by following the trajectories from large T to small T.

Therefore, we apply "sub-goal tree (SGT)" (Jurgenson et al., 2020b) to recursively divide a trajectory from small T to large T and produce a sub-task tree. We define a planning policy $\pi_p(g|g_i, g_j)$ as a stochastic mapping from two nearby endpoints g_i and g_j to a predicted sub-goal g in the middle of g_i and g_j. In our scheme, we use $\pi_p(g|g_i, g_j)$ to break down a task with initial state g_i and goal g_j (denoted by (g_i, g_j)) to two sub-tasks (g_i, g) and (g, g_j). Hence, we can generate a tree of sub-goals by recursively sampling sub-goals from $\pi_p(g|g_i, g_j)$ as below, which finally generates a planning trajectory $g_{0:T}$ with a tree structure, i.e.,

$$\Pr_{\pi_p}(g_{0:T}|g_0 = s_0, g_T = g) \triangleq \Pr_{\pi_p}\left(g_{0:\frac{T}{2}} \middle| g_0, g_{\frac{T}{2}}\right) \Pr_{\pi_p}\left(g_{\frac{T}{2}:T} \middle| g_{\frac{T}{2}}, g\right) \pi_p\left(g_{\frac{T}{2}} \middle| s_0, g\right), \quad (3.7)$$

where $T = 2^K$ with K being the depth of the tree. As shown in Figure 1.2, for layer-k, the sub-goal tree $g_{0:T}$ interpolates a sequence of $2^k - 1$ sub-goals $g^k_{1:(2^k-1)} \triangleq \left(g^k_1, g^k_2, \ldots, g^k_{2^k-1}\right)$ between s_0 and g, where $g^k_j = g_{Tj/2^k}$ in $g_{0:T}$, $\forall\ j \in [2^k - 1]$.

In layer-1, we have the coarsest trajectory $(s_0, g^1_1 = g_{T/2}, g)$. In the bottom layer-$K$, we have the finest trajectory $g_{0:T}$. From top layers to bottome ones, their sub-goal trajectories naturally form a coarse-to-fine sub-tasking curriculum, e.g., the planning in layer-1 requires the agent to accomplish two hard and long-horizon sub-tasks to reach g, while layer-K's planning requires the agent to accomplish T much simpler and shorter-horizon sub-tasks.

To train the planning policy π_p, we apply it to produce a tree-structured $g_{0:T}$ via Eq. (3.7) and evaluate the cost $c\left(g^k_{0:2^k}\right)$ of the trajectory $g^k_{0:2^k}$ by integrating the cost of every segment/sub-task $c\left(g^k_{tT/2^k}, g^k_{(t+1)T/2^k}\right)$ along the trajectory. We will elaborate on our option of cost function $c(\cdot, \cdot)$ later in Eq. (3.11). The objective of π_p aims to minimize the total cost $c(g_{0:T})$ of the sub-goal tree, which sums over all

trajectories' costs across the K layers,

$$c(g_{0:T}) \triangleq \sum_{k=0}^{K} c\left(g^k_{0:2^k}\right). \tag{3.8}$$

According to APSP objective in Eq. (3.5), the optimal planning policy π_p^* minimizes the expected cost J_{π_p} over all possible planning trajectories defined below:

$$J_{\pi_p} \triangleq \mathbb{E}_{g_{0:T}}[c(g_{0:T})] = \mathbb{E}_{(s_0,g)}\mathbb{E}_{g_{1:T-1}\sim\pi_p}[c(g_{0:T})], \tag{3.9}$$

where $g_{1:T-1} \sim \pi_p$ denotes the recursive sampling of $g_{1:T-1}$ in Eq. (3.7). Any policy gradient method can be used to minimize J_{π_p}, with the gradient w.r.t. π_p computed as

$$\nabla J_{\pi_p} = \mathbb{E}_{g_{0:T}\sim\pi_p}\left[c(g_{0:T}) \cdot \nabla \log \Pr_{\pi_p}(g_{0:T}|s_0, g)\right]. \tag{3.10}$$

To form an easy-to-hard curriculum for training π_p, during the top-down construction of the tree, at every layer-k, we train π_p to only minimize the cost for layers from 0 to k (instead of K as in Eq. (3.8)) so the planning policy π_p starts from only producing relatively coarse sequence of a few sub-tasks for the top layers before trained to produce more detailed sub-task paths. In Line 4 of Algorithm 0, we will use $J^k_{\pi_p}$ to denote $c(g_{0:T})$ computed up to layer-k.

Cost function of sub-tasks: As discussed in the end of Section 3.1.2, the cost function $c(g_t, g_{t+1})$ should reflect the difficulty of sub-task (g_t, g_{t+1}) for the RL agent. Euclidean distance $\|g_t - g_{t+1}\|_2$ is commonly used by previous path-planning methods but is not adaptive to the evolution of the agent's policy and environment, e.g., the difficulty of sub-task (g_t, g_{t+1}) with/without nearby obstacles can vary drastically. Instead, we use the time-cost $\tau_{g,g'}$ spent by the agent on completing the task (g, g') to measure its difficulty, which is adaptive to both the agent and environment and thus more accurate than Euclidean distance. By training the planner to produce minimum-cost sequences of sub-tasks, the planned paths are optimized for the training of RL policy.

Algorithm 1 Top-Down Construction of Sub-Task Tree

Input: (s_0, g), planning policy π_p and its training set $\mathcal{D}_\tau$

Output: tree structured sub-goals $g_{0:T}$, π_p

for $k = 1, 2, \ldots, K$ **do**

    Apply any RL method to minimize $J^k_{\pi_p}$, i.e., J_{π_p} in Eq. (3.9) computed only up to layer-k;

    for $t = 0, 1, \ldots, 2^{k-1} - 1$ **do**

        Generate the sub-goal $g^k_t \sim \pi_p(g^k_t | g^{k-1}_{t-1}, g^{k-1}_t)$;

        Add g^k_t, g^{k-1}_t into the trajectory $g^k_{0:T}$ on layer-k;

    end for

end for

Since the time cost data is collected during the training of RL agent on the assigned sub-tasks, they are not available at the very beginning of the first episode. Therefore, we "warm start" the first top-down construction and training of the planner by Euclidean distance and consider the following cost function for the first two episodes, i.e.,

$$c(g_t, g_{t+1}) = \alpha \|g_t - g_{t+1}\|_2 + (1 - \alpha)\tau(g_t, g_{t+1}). \tag{3.11}$$

In experiments, we start from $\alpha = 1$ and gradually reduce it towards 0 during the first two episodes. After that, the cost function is the time cost $\tau(g_t, g_{t+1})$ collected in the previous episode and no longer depends on the Euclidean distance.

3.3 CO-PILOT Algorithm

CO-PILOT is a mutual training scheme between the RL policy π and the planning policy π_p, each generating dense cost/reward on tree-structured sub-tasks to train the other. **By top-down construction of sub-task tree from $k = 0$ to $k = K$,** it firstly trains the planning policy π_p on a curriculum of generating coarse-to-fine

trajectories. On each layer-k, it generates 2^k sub-tasks through rollouts of π_p. Given $\mathcal{D}_\tau$ and the cost $c(\cdot,\cdot)$ in Eq. (3.11), CO-PILOT updates π_p by minimizing J_{π_p} in Eq. (3.9). At the very beginning of CO-PILOT, $\mathcal{D}_\tau = \emptyset$ and the cost solely depends on the Euclidean distance. However, as we are collecting more experiences into $\mathcal{D}_\tau$, π_p will be trained towards producing the easiest sub-task trajectory for the RL agent to finish and thus increases its chance of receiving non-zero rewards. The complete procedures of top-down construction are given in Algorithm 0. Being updated using the most recent $\mathcal{D}_\tau$, π_p keeps tracking the RL agent's learning progress to produce the most cost-efficient paths for the agent. Moreover, the top-down construction naturally forms an **e**asy-to-hard curriculum for the planning policy π_p. In Line 2 of Algorithm 0 , we train π_p to produce sub-goal trajectories up to layer-k. Hence, the training of π_p is more smooth and less challenging than learning the optimal V^* or Q^*.

After the top-down construction of the sub-task tree, CO-PILOT trains the **RL policy π on a curriculum of easy-to-hard** sub-tasks by **b**ottom-up traversal of the tree from $k = K$ to $k = 0$. The sub-goal trajectory in each layer aims to guild the agent to complete the original task from s_0 to g. At layer-k, CO-PILOT applies π sequentially to the 2^k sub-tasks (as the conditioned goal). It then updates π by SAC, which alternates among the minimization of J_Q, J_π and J_V in Eq. (3.1)-(3.4) to update π, V and Q.

Note we can replace SAC with other RL algorithms in the general framework of CO-PILOT. The rollouts of π on the sub-tasks not only collect experiences to train itself but also collect tuples of $(g, g', \tau_{g,g'})$ added to $\mathcal{D}_\tau$, which will be used to train π_p. It is possible that π fail on some sub-task within $\tau_{\max}$ steps. In this case, we treat the actual ending state as g' in the tuple for $D\tau$ and initialize the the next sub-task from this state. The bottom-up traversal is detailed in Algorithm 2, where Line 14-18 apply π to reach sub-goal $g_{tT/2^k}$. The bottom-up traversal forms an easy-to-hard curriculum to train π, in which the sub-tasks from the bottom layers

Algorithm 2 Bottom-Up Traversal of Sub-Task Tree

Input: RL policy π, sub-goal tree of $g_{0:T}$, $\tau_{\max}$, ϵ

Output: π, $\mathcal{D}_\tau$

Initialize: $\mathcal{D}_\tau \leftarrow \emptyset$

for $k = K, \ldots, 1, 0$ **do**

  Set RL agent's initial state to be $s_0 \leftarrow g_0$;

  for $t = 1, 2, 3, \ldots, 2^k$ **do**

    Set the condition of V, Q, π in SAC to be g_t^k;

    $\tau \leftarrow 0, \quad \mathcal{B} \leftarrow \emptyset$;

    while $\tau \leq \tau_{\max}$ or $s_\tau \notin B(g_t^k, \epsilon)$ **do**

      RL agent takes action $a_\tau \sim \pi(a_\tau | s_\tau, g_t^k)$;

      RL agent moves to $s_{\tau+1} \sim p(s_{\tau+1} | s_\tau, a_\tau)$ and receives reward $r(s_\tau, a_\tau | g_t^k)$;

      $\mathcal{B} \leftarrow \mathcal{B} \cup (s_\tau, a_\tau, r(s_\tau, a_\tau | g_t^k), s_{\tau+1})$;

    end while

    $\mathcal{D}_\tau \leftarrow \mathcal{D}_\tau \cup (s_0, s_\tau, \tau), \quad s_0 \leftarrow s_\tau$;

    for every gradient step **do**

      Apply gradient steps in SAC: update Q, V, π to minimize J_Q, J_π and J_V in Eq. (3.1)-(3.4) using samples drawn from $\mathcal{B}$;

    end for

  end for

end for

Algorithm 3 CO-PILOT

Input: $\mathcal{G}$, p_0, T, $\tau_{\max}$, ϵ, b

Output: RL agent's policy π, planning policy π_p

Initialize: π, π_p, $\mathcal{D}_\tau$ by Euclidean distance

while not converge **do**

    Sample a task (s_0, g) with $s_0 \sim p_0(s)$ and $g \in \mathcal{G}$;

    for episode $= 1, 2, \ldots, b$ **do**

        Algorithm 0: top-down construction of a sub-task tree $g_{0:T}$, train planning policy π_p based on $\mathcal{D}_\tau$;

        Algorithm 2: bottom-up traversal of the sub-task tree $g_{0:T}$, train RL policy π, collect $\mathcal{D}_\tau$;

    end for

end while

are easier so the agent by larger chance can receive non-zero rewards. Given a task (s_0, g), the curriculum guides the agent first to learn how to finish it by following a detailed planning path of many sub-goals. It then gradually increases the hardness by halving the number of sub-goals until recovering the original task. Therefore, it critically alleviates the sparse reward problem that usually fails or considerably slows down RL on long-horizon tasks.

A prominent advantage and difference of CO-PILOT compared to other methods that combine RL and planning is to repeat the top-down construction and bottom-up traversal for multiple (b in Algorithm 3) episodes on each task (s_0, g). Thereby, the RL agent and the planning policy are fully optimized for each other's training, forming an adaptive curriculum without human engineering. The complete procedures of CO-PILOT are listed in Algorithm 3.

3.4 Empirical Evaluation

We evaluate CO-PILOT on three types of tasks: a maze and two continuous control tasks for robotic navigation. CO-PILOT outperforms several strong baselines of RL and planning, as well as methods combining RL and planning, on both sample efficiency and final success rate.

3.4.1 Environment Setup

Maze environment: We build a maze environment of size 1×1 containing square obstacles(obstacle states) and free-to-reach states as shown in Figure 3.4. For each benchmark(the design of benchmark refer to the caption of Figure 3.1), We randomly sample 300 pairs of (s_0, g) for training and 100 pairs for test from a uniform distribution on the coordinate range and remove the ones in obstacles. It takes an RL agent $\geq$ 200 steps on average moving from s_0 to g, which is a long-horizon task. The task succeeds if the agent reaches $B(g, \epsilon = 0.025)$ without collision. We design three benchmark environments in Maze (as shown in Figure 3.2). Three environments differ in the choice of robots:

- **Workspace planning (2d):** The robot is abstracted with a point mass moving in the plane. Without higher dimensions, this problem reduces to planning in the workspace.
- **Rigid body navigation (3d):** A rigid body robot, abstracted as a thin rectangle, is used here. This robot can rotate and move freely without any constraints in the free space.
- **3-link snake (5d):** The robot is a 5 DoF snake with two joints.To prevent links from folding, we restrict the angles to the range of $[-\pi/4, \pi/4]$.

Mujoco Ant-v1: We evaluate CO-PILOT and baselines in the Mujoco environ-

ment with an Ant-v1 agent (Quadruped Tassa et al. (2018)) with an 8-dim action space. As shown in Figure 3.1 (c), we train the agent to navigate in the maze without self-rotation and collisions to the wall. We randomly sample 50 (s_0, g) pairs for training and 10 pairs for test.

BipedalWalker: The BipedalWalker environment (Brockman et al., 2016) offers a new perspective of tasks rather than maze type. The learning agent, embodied in a bipedal walker, receives positive rewards for moving forward and penalties for torque usage and angular head movements. Agents are allowed 2000 steps to reach the other side of the map. The environment producing tracks paved with stumps varying by their height parameter μ_h and an independent sampled spacing parameter $\Delta_s \in \mathcal{N}(\mu_h, 0.1)$. We design three agents with different length of legs (as shown in figure 3.3) for each benchmark. Performance is evaluated periodically by sampling 10 tracks in each track distribution of a fixed evaluation set of 50 distributions sampled uniformly in the parameter space. We measure the percentage of mastered tracks.

Baselines: In the maze environment, we compare CO-PILOT with (1) three planning methods: RRT* (Karaman and Frazzoli, 2011) (Rapidly-exploring Random Trees), NEXT (Chen et al., 2020a) (Neural Exploration-Exploitation Trees) and SGTPG (Jurgenson et al., 2020b) (Sub-Goal Tree Policy Gradient); three model-free RL algorithms: valued-based method SAC (Haarnoja et al., 2018b), policy-based method PPO (Schulman et al., 2017) (in CO-PILOT, we use the former to train the RL policy and the latter to train the planning policy) and HER (Andrychowicz et al., 2017b), which improves goal-conditioned RL's efficiency by re-labelling the visited states as pseudo goals; and (3) a RL-planning hybrid method: SoRB (Eysenbach et al., 2019a), which trains planning strategies based on the experiences of a given RL policy. For fair comparisons, we use SAC as the RL algorithm in both CO-PILOT and SoRB. In the Mujoco environment, we compare CO-PILOT with SAC, SoRB, and hierarchical RL (Hejna et al., 2020). In BipedalWalker, we

compare CO-PILOT with SAC and SoRB.

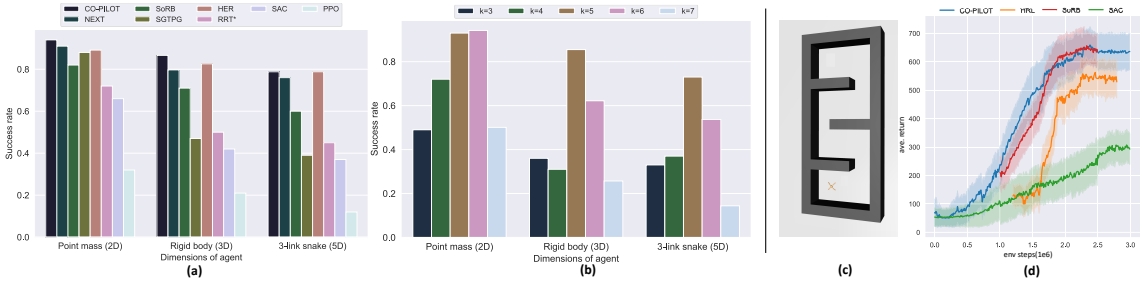

Figure 3.1 : **(a)** Success rate on test tasks of Maze environment. We train three types of agents with different DoF (degrees of freedom): point mass (2D), rigid body (3D), and 3-link snake (5D). **(b)** Success rate of CO-PILOT with sub-goal tree of different depth K on the same test set in **(a)**. **(c)** Ant-v1 agent and the associated environment in Mujoco. **(d)** Average return of Ant-v1 in **(c)**.

3.4.2 Training Details and Hyperparameters

In CO-PILOT, we initialize the dataset $\mathcal{D}_\tau$ with 50,000 tuples of $(g, g', \tau_{g,g'})$ with $\tau_{g,g'}$ being the Euclidean distance. We use SAC to train the RL policy and PPO (Schulman et al., 2017) to train the planning policy in Line 8 of Algorithm 0, since the former encourages exploration and the latter is simple and efficient. We set a reward of 1 $(1000, 200)$ to each task (s_0, g) in Maze (Mujoco,BipedalWalker). According to Eq. (3.6), the reward of each sub-task in layer-k is $1/2^k$ $(1000/2^k, 200/2^k)$. For planning cost, if the segment between (g_t, g_{t+1}) trespasses any obstacle, we add a penalty of 10 to $\tau(g_t, g_{t+1})$ in Eq. (3.11). We linearly reduce α in Eq. (3.11) from 0.9 to 0.1 throughout every episode. For planning policy training, we apply PPO with a trust region of $\epsilon = 0.2$ and use Adam optimizer (Kingma and Ba, 2015) with a learning rate of 0.005. For RL training with SAC, we use its default hyperparameters. In both environment, we set $T = 2^5$ (ablation study of different T in Figure 3.1 (c)) and $b = 5$ (further increasing it does not improve the performance). We set

$\tau_{\max} = 25, \tau_{\max} = 200$ and $\tau_{max} = 2000$ for Maze, Mujoco and BipedalWalker respectively. For efficiency, in Line 5 of Algorithm 3, we instead sample a mini-batch of 30 (Maze) or 50 (Mujoco) pairs of (s_0, g). The (s_0, g) is fixed in BipedalWalker, we randomly sample 20 tracks in each track distribution from the same 50 distributions mentioned in section 3.4.1.

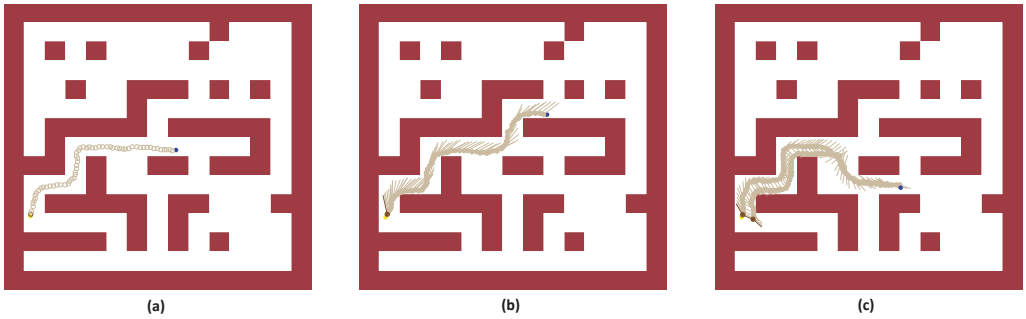

Figure 3.2 : Three different agents in Maze environment. (a) 2d workspace, (b) Rigid body, (c) 3-link snake. The deep brown agent shows when the agent is around the initial state. The light brown agent shows the actual RL agent navigation trajectory. Yellow point and blue point represent for the initial state s_0 and goal state g respectively.

3.4.3 Hyperparameters in CO-PILOT

The settings of the hyperparameters are known as the key for the success of an RL algorithm. Thus, we list the hyerparameters of CO-PILOT in Maze environment and continuous control environment in table 3.1 and table 3.2 respectively.

3.4.4 Main Results

In Figure 3.1 (a), we compare the performance of CO-PILOT with all the baselines on the test tasks of the Maze environment. CO-PILOT achieves the highest success rate across all benchmarks and significantly outperforms SAC and SGTPG. Figure 3.4 (a)-(c) report how the success rate of all methods change during training

Table 3.1 : Hyperparameter of SAC in Maze environments

Parameter	Value
optimizer	Adam
Timesteps	1.8×10^6
learning rate	$3 \cdot 10^{-4}$
discount (γ)	0.99
replay buffer size	10^6
number of hidden layers (all networks)	2
number of hidden units per layer	256
number of samples per minibatch	200
nonlinearity	ReLU
target smoothing coefficient (τ)	0.005
target update interval	1
gradient steps	1

as the number of interaction steps with the environment increases. We limit the total environment steps of all methods $\leq 1.8 \times 10^6$ except for NEXT (since NEXT requires the pre-training of RRT*). In Figure 3.4 (a), CO-PILOT and SGTPG perform similarly in the early training period because the cost data collected by the RL agent do not contain sufficient information to train a powerful path-planner and the inaccurate Euclidean distance dominates the cost $c(\cdot, \cdot)$ in Eq. (3.11). The performance of SoRB and SAC are similar because SoRB needs to pre-train the RL policy at first when no planning is required. SoRB surpasses SAC during the later stages. NEXT also needs to pre-train RRT* before applying self-improving training, so we do not see the change of cost and collision checks for NEXT during the earlier stages. The comparison between CO-PILOT with SAC demonstrates the significant

Table 3.2 : Hyperparameter of SAC in continuous control environments

Parameter	Value
optimizer	Adam
Timesteps	3×10^6
learning rate	$5 \cdot 10^{-4}$
discount (γ)	0.99
replay buffer size	10^6
number of hidden layers (all networks)	2
number of hidden units per layer	256
number of samples per minibatch	256
nonlinearity	ReLU
target smoothing coefficient (τ)	0.005
target update interval	1
gradient steps	1

improvement brought by the learned planner to RL.

The average return on Mujoco tasks is shown in Figure 3.1 (d). Furthermore, the percentage of mastered environments on BipedalWalker is shown in Figure 3.3. For simplicity, we denote Hejna et al. (2020) as HRL. SoRB and HRL start later because the RL policy is under pre-training. The experimental results show that CO-PILOT achieves much better sample efficiency than all the baselines, including SoRB. The final performance is comparable with SoRB but significantly outperforms SAC and HRL(Mujoco).

CO-PILOT with sub-task tree of different depth K: In Figure 3.1 (b), we evaluate 5 different depths for the sub-task tree in CO-PILOT, with all the rest

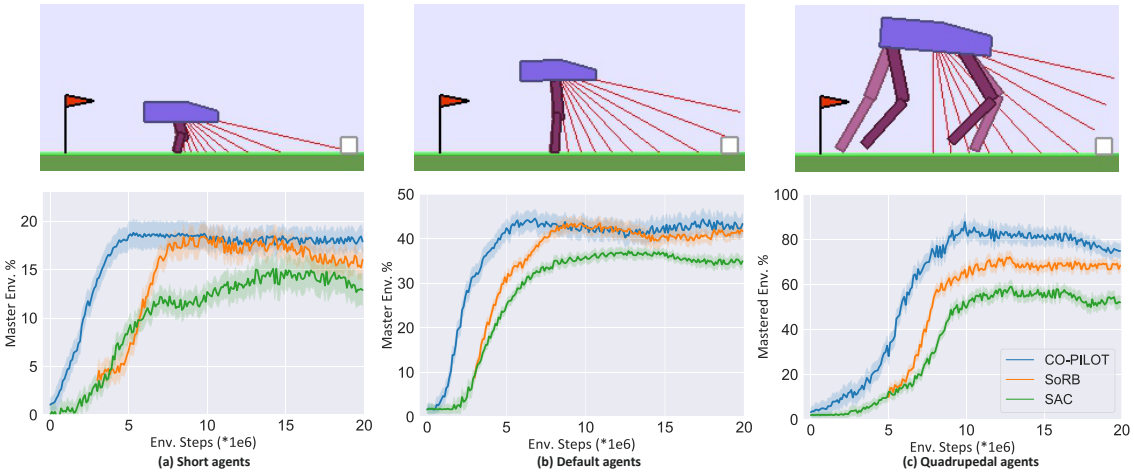

Figure 3.3 : **Main results and comparison in BipedalWalker.** The mean performance (32 seeded runs) is reported together with the standard deviation (shaded areas).

hyperparameters fixed. It shows a trade-off between RL and planning, i.e., a deeper sub-task tree can provide denser rewards and more detailed guidance from sub-tasks, hence improving the efficiency of RL, but it also makes training the planning policy more challenging. In this experiment, the best trade-off is achieved when depth $K = 5$.

3.4.5 Case Study: A Close Look of Mutual Training in CO-PILOT

To understand the mutual training of CO-PILOT in the experiments, in Figure 3.4 (d), we visualize how a sub-task tree evolves over episodes in Algorithm 3, where the sub-goal paths at layer-$2, 3, 4$ generated in Episode-$1, 3, 5$ for the same task (s_0, g) are reported. Each maze map contains a path from a layer in an episode and the histogram above it reports the time cost of RL for completing each sub-task on the path. We are particularly interested in two questions: how does planning guide RL by sub-tasking? How does RL agent's cost affect the planned paths?

Train π_p using RL agent's time cost data. In Episode-1, the sub-goal paths of all the three layers are too close to some obstacles or even trespass some others and thus cannot provide reliable guidance for the RL agent. The Euclidean distances between consecutive sub-tasks on a path are almost equal but the corresponding time costs shown in the histograms vary a lot, which is not preferred since some sub-tasks are too hard, but some are too easy for training the RL agent. Hence, the planning policy is not fully optimized to produce cost-efficient paths for the RL agent.

In Episode-3, the generated sub-goals paths become more adaptive to the environment. In all layers, we can see that the planner tends to generate longer segments for places with fewer nearby obstacles and collision risks while adding more fine-grained sub-tasks to get around the corners. This phenomenon implies that the planning policy is learning to produce better and more adaptive guidance with dense rewards. However, due to the limited number of sub-goals per layer, the paths in layer-$2, 3$ can still be improved if interpolating more sub-goals. Nevertheless, on the deepest layer-3, the planned path is already collision-free and thus can provide an accurate reward shaping for the RL agent.

In Episode-5, the planning paths are almost optimal, especially for the one in layer-4, which keeps distant from the obstacles of both sides in the maze. Moreover, RL agent's time costs shown in the histograms are not only much lower than those of the previous two episodes but also have similar values across different sub-tasks. Hence, the planning policy is well optimized to generate cost-efficient paths for RL.

Planner guides RL by sub-tasking. In Episode-1, the time cost for the RL agent to finish the whole task is much higher than that in the later episodes due to the poor RL policy at the beginning. In contrast, the time cost drastically decreases in Episode-3 when compared to Episode-1, which indicates that the RL policy is

significantly improved under the guidance of planning. In Episode-5, the time cost for completing each sub-task of layer-$3, 4$ further decreases when compared to the former episodes and the costs becoming more uniform across sub-tasks, implying that the RL policy learns to complete the planned sub-tasks more efficiently.

Therefore, both planning and RL are improved via the mutual training scheme in CO-PILOT and fully optimized to facilitate the training of the other. In particular, the planner learns to produce cost-efficient paths of different amounts of sub-tasks to guide RL with dense rewards. It does not depends on any pre-defined metric but is adaptive to the RL policy. Moreover, it does not need to produce a step-by-step plan: the RL agent can learn efficiency under the guidance of a few interpolated sub-tasks by the planner. Hence, the mutual training between the two policies overcome the bottlenecks of training each policy separately. In addition, the easy-to-hard curricula for both planning and RL considerably improves their training efficiency.

3.5 Discussion

In this chapter, we introduce CO-PILOT, a mutual learning framework between RL and "learning to plan" policies, which provides a principal solution addressing the problems of both RL and planning when applied to long-horizon tasks. In CO-PILOT, each policy produces dense feedback on a curriculum of sub-tasks to train the other more efficiently and is optimized to assist the other's training. The planner learns to decompose a long-horizon task into a few sub-tasks at first and then gradually increases the interpolated sub-tasks, forming an easy-to-hard curriculum to train the planning policy. On the other hand, this top-down curriculum recursively builds coarse-to-fine sequences of sub-tasks. By training the RL agent to complete easier sub-tasks on finer sequences of bottom layers at first and then gradually moving to harder ones in top layers, the RL agent can be efficiently trained following an easy-to-hard curriculum. In experiments, CO-PILOT significantly im-

proves the sample efficiency and success rate on different types of tasks especially on long-horizon tasks with sparse rewards.

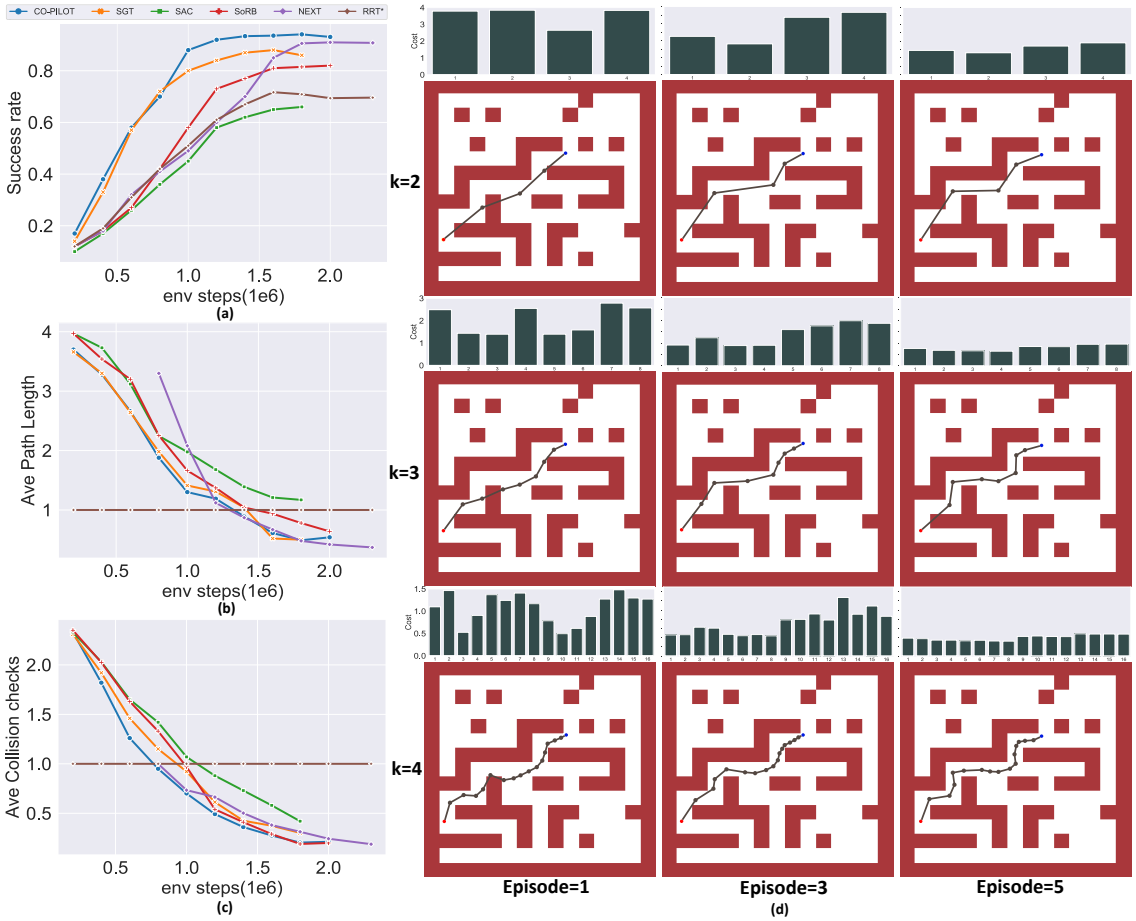

Figure 3.4 : **Case study results.** **(a)** Success rate. **(b)** Average path length (normalized by RRT*) in terms of Euclidean distance. **(c)** Average collision checks (normalized by RRT*) as every method increases its interaction steps with the environment. CO-PILOT achieves the best sample efficiency among all methods. **(d)** Visualization of the sub-goal paths on layer-k $= 2, 3, 4$ of the sub-task tree in Episode-$1, 3, 5$ for a task with initial state s_0 (red dot) and goal g (blue dot) in Maze. Each histogram reports the RL agent's cost $\tau_{g_t, g_{t+1}}$ for sub-tasks along the path. As episode increases, planning paths across all layers are improved, and on each path the costs of all sub-tasks reduce towards a similar value, though the Euclidean distances are still different, since the planner learned to produce more sub-tasks near complicated obstacles.

Chapter 4

Adversarially Generated Environments for RL

4.1 Basic Settings

We recall the collaboratively training scheme introduced in sec. 3.1 of Chapter. 3. Given a 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 Eq. 3.10. 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 following Eq. 3.7

4.1.1 Adversarial-Environment Generator (EG) applied to each Sub-task

Assume a k-level sub-task tree at time t, 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. 3.1.1, the learning objective of EG is to maximize its expected return over different tasks (s_0, g), i.e., $\max_{\pi_e} \mathbb{E}_{(s_0,g)}[\mathbb{E}_{\pi_e}(R_0^e)]$ where the return is defined as $R_t^e = \sum_{k=1}^{K} \sum_{i=t}^{2^k} \gamma_e^{i-t} r_e(s_t^k, E_t^k | g)$ 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. 3.1.1, we can apply any RL algorithm to train EG, e.g., we use A2C (Mnih et al., 2016) for the experiments.

4.2 Environment Adversarial sub-Task Tree Curriculum (EAT-C)

4.2.1 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.

4.2.2 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 0. 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 5), 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 4 Top-Down Planning of Sub-task Curriculum

1: **Input:** (s_0, g), T, planning policy π_p and its training set $\mathcal{D}_p$.
2: **Output:** tree structured sub-goals $g_{0:T}$, π_p
3: **for** $k = 1, 2, \ldots, K$ **do**
4: Apply an RL algorithm to minimize J_{π_p} in Eq. (3.7) 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**

Bottom-Up Curriculum for RL and EG Given a sub-task tree generated by Algorithm 0, EAT-C trains RL policy π and EG policy π_e by following a bottom-up traversal of the tree. As shown in Algorithm 5, 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 $g^k_{0:2^k-1}$ 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.

EAT-C algorithm The complete procedure of EAT-C is introduced in Algorithm 6. Given a long-horizon task (s_0, g) (line 5), EAT-C iterates between the top-down and bottom-up procedures in Algorithm 0-2 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.

4.3 Empirical Evaluation

4.3.1 Experiment Setup

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

Algorithm 5 Bottom-Up traversal in EAT-C

1: **Input:** RL and EG policy π, π_e, sub-goal tree $g_{0:T}$, $\tau_{\max}$, ϵ

2: **Output:** π, π_e, $\mathcal{D}_p$

3: **Initialize:** π_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 modifies the environment $\mathbf{E}^k_{t-1}$ to $\mathbf{E}^k_t$;

8: Apply RL agent to complete sub-task (g^k_{t-1}, g^k_t), and store $(s_\tau, a_\tau, r(s_\tau, a_\tau | g^k_t), s_{\tau+1})$ to $\mathcal{D}(\tau \leqslant \tau_{\max})$;

9: REACH(s, g^k_t);

10: **end for**

11: Update π and π_e using samples in $\mathcal{D}$;

12: **end for**

13: **Procedure** REACH(s, g):

14: **if** $d(s, g) \leq \epsilon$ **then**

15: $\mathcal{D}_e \leftarrow \mathcal{D}_e \cup (s_\tau, b_t, r_e(s_\tau, E^k_t | g), g^k_t)$;

16: $\mathcal{D}_p \leftarrow \mathcal{D}_p \cup (s_0, s_\tau, \tau), \quad s_0 \leftarrow s_\tau$;

17: **else**

18: Re-apply π_p to interpolate temporary sub-goals for (g^k_{t-1}, g^k_t);

19: Repeat Line.8 with new generated sub-goal sequence;

20: REACH(s, g);

21: **end if**

Algorithm 6 EAT-C

Input: p_0, T, $\tau_{\max}$, ϵ, n
Output: π, π_p, π_e
Initialize: π, π_p, $\mathcal{D}_p$
while not converge **do**
    Sample a task (s_0, g) with $s_0 \sim p_0$ and $g \in \mathcal{G}$;
    for episode $= 1, 2, \ldots, n$ **do**
        Algorithm 0: top-down construction of a sub-task tree $g_{0:T}$, train planning policy π_p based on $\mathcal{D}_p$;
        Algorithm 2: bottom-up traversal of the sub-task tree $g_{0:T}$, train RL policy π and EG policy π_e, collect $\mathcal{D}_p$;
    end for
end while

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. 4.5, 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 and Pal, 2018). We train and test RL policies on three types of compositional tasks depicted in fig 4.3, i.e., hunting, scavenging, and salad-making, as illustrated in Fig. 4.3.(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. (2020a)) 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. 4.1).

Baselines: We compare EAT-C with a broad class of RL methods having related ideas to EAT-C: (1) ALP-GMM (Portelas et al., 2019b) that generates a curriculum of diverse tasks with large progress for goal-conditioned RL; (2) POET (Wang et al., 2019b) 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.

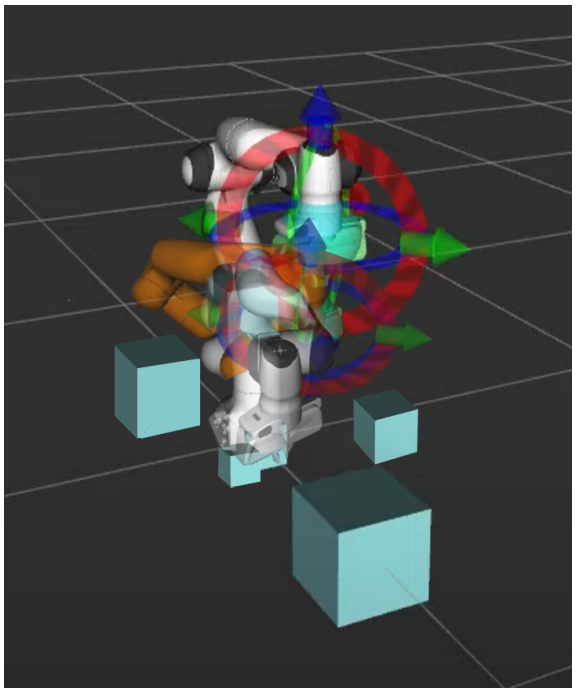

Figure 4.1 : 7DoF Robotic Arm in a training environment with randomly sampled obstacles (those cyan cubes).

4.3.2 Detailed Evironment settings in EAT-C

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. 4.1. 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

$$b_t \triangleq \Delta\theta_i = (\Delta x_i, \Delta y_i, \Delta w_i, \Delta h_i), \quad \forall i \in [n]. \tag{4.1}$$

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

$$\Delta x_i \leftarrow \min\{\max\{\Delta x_i, \beta_x \cdot x_{\min}\}, \beta_x \cdot x_{\max}\}, \tag{4.2}$$

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.

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.

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.

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 4.1, 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 4.1 : 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.

4.3.3 Model Architecture and Hyperparameters in EAT-C

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 4.2. 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. 4.3.

Table 4.2 : 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	$-\dim(\mathcal{A})$

4.3.4 Main Results

We report the performance of EAT-C and all baselines evaluated on the test tasks in Fig. 4.2(a) for 2D pusher, in Fig. 4.3(d)-(f) for the discrete space tasks and in Table. 4.4. In all experiments, EAT-C outperforms all other baselines by a large

Table 4.3 : 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	$-\dim(\mathcal{A})$

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. 4.4 indicates that EAT-C improves the RL policy in complex tasks.

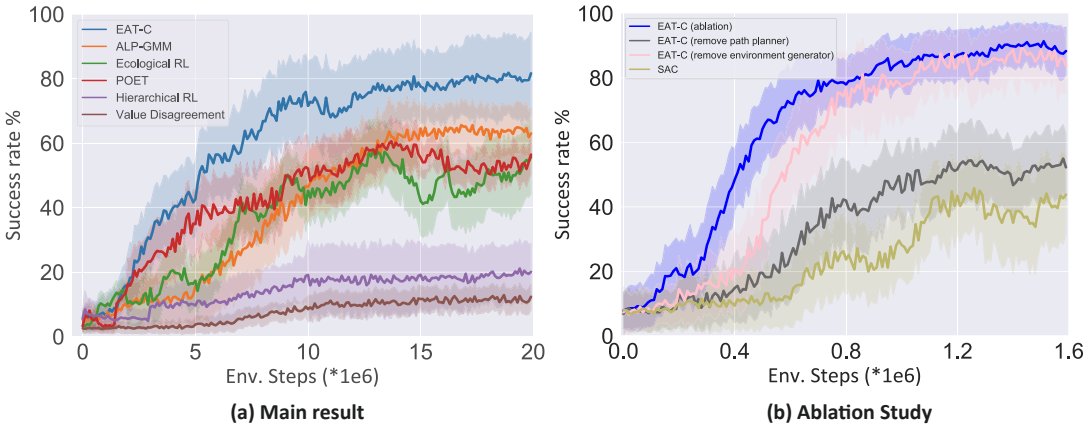

Figure 4.2 : **(a)** report the success rate (mean±std averaged over 6 random seeds) of EAT-C and baselines on test tasks in 2D Pusher environments. **(b)** Ablation study of EAT-C on 2D Pusher tasks.

Among the three compositional tasks in Fig. 4.3, 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.

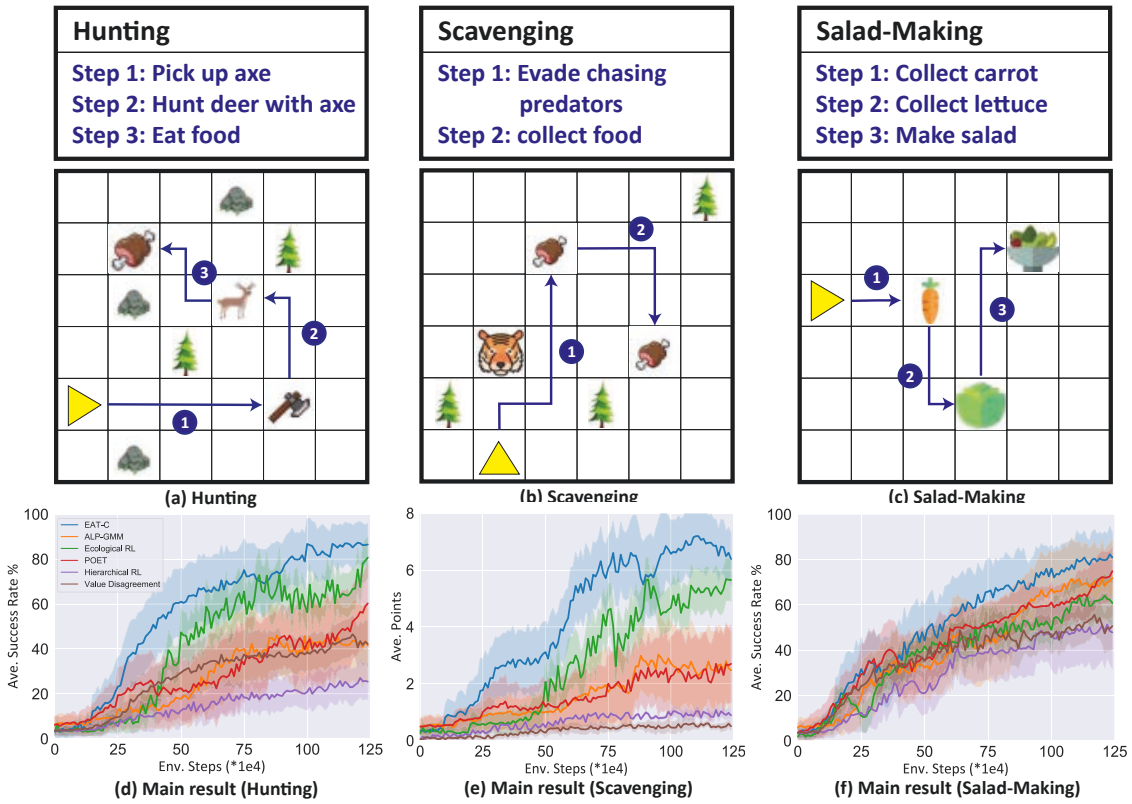

Figure 4.3 : **(a)-(c)** illustrate the 2-3 key steps for completing each task. In Scavenging, the agent will have 2 points when it collects food each time. **(d)-(f)** report different methods' performance (mean±std over 10 random seeds) on multiple test tasks.

Methods	Average Collision Rate	Success Rate
EAT-C	**0.22** ± 0.05	**0.873** ± 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 ± **0.02**	0.772 ± **0.014**

Table 4.4 : 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.

Analysis on main results. Refer to the results reported in Fig 4.2, Fig 4.3, and Table 4.4, 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.

4.3.5 Ablation study

Albation study - components in EAT-C. 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. 4.2(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. 4.5.

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 4.5 : Success rate on test tasks in random environments and training environment. Different from the ablation study in Fig. 4.2(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.4.2(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. 4.5). 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. 4.2(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. 4.5). 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.

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. 4.2(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. 4.2(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.

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. 4.4, we report the

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 4.6 : Ablation Study Results (larger version of Table. 4.5)

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.

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. 2) and baseline methods. The connectivity test is common in navigation as well as many complicated and realistic environments/tasks (Francis et al., 2020; Yingjun and Xin-wen, 2019). For example, in maze tasks like 2D-pusher, modifying the obstacles easily results in unsolvable environments. Whether to apply

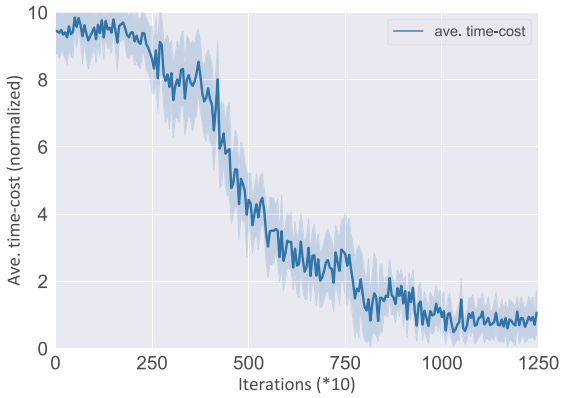

Figure 4.4 : Average time-cost of the RL agent to complete a sub-task from layer-3 of the sub-task tree. As the training proceed, time-cost that the agent needs to complete each sub-task decreases significantly, indicating that π_p does not propose infeasible goals, and EG does not make the reward more sparse.

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 test is not required by any method. However, as we show in the table 4.7, ALP-GMM and POET generate more unsolvable environments than EAT-C when applied to 2D-pusher.

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

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 4.7 : 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.

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.

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. 4.8. 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 → POET)" in table 4.8. Note that "EAT-C (EG → 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	**2.5** million steps	**5.0** million steps	**7.5** million steps	**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 → 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 4.8 : In this experiment, we compare the performance of training RL policy in diverse environments mutated by POET (EAT-C (EG →POET)) with the original EAT-C and POET. Note that EAT-C (EG → 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 → 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 subtask curriculum with adversarial environments is more efficient than some existing curricula applied to HRL.

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 4.9. 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	**5.0** million steps	**10.0** million steps	**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 4.9 : 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.

4.3.6 An Emprical Study: How does EAT-C work?

To better understand how the path planner and EG help RL in EAT-C, in Fig. 4.5, 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. 6) 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

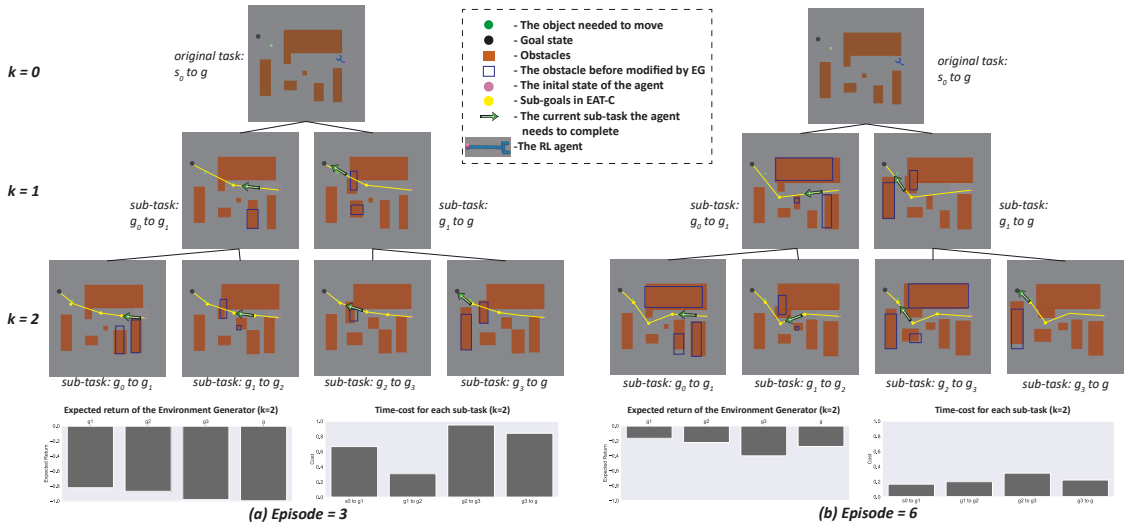

Figure 4.5 : Visualization of EAT-C. A 2D robot with a 4-joint arm starts from the initial state (pink), navigates to the object (green) location, and then pushes the object to the goal state (black). The histograms in (a) and (b) represents the expected return of EG taking action b_t, and the costs of sub-tasks predicted by the path planner in layer $k = 2$, respectively.

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. 4.5, 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.

4.4 Discussion

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.

Chapter 5

Morphology-Environment Co-Evolution Framework

5.1 Preliminaries and Basic Settings

In this chapter, we investigate the problem of improving the RL agent's morphology to be adaptive to diverse environments and we propose a novel framework "Morphology-Environment Co-Evolution (MECE)". In MECE, we have three RL policies, a control policy π that learns to control the agents of evolved morphology, a morphology policy π_m that learns to modify the agent's morphology for better robustness in diverse environments, and an environment policy π_e that learns to change the environment to boost the morphology evolution. During the training phase, π_m and π_e are alternately applied to evolve the agent's morphology and the training environment, respectively. Taking into account that π might require various environment steps for training in distinct morphologies or environments, we propose a dynamic time-scaling for π_m and π_e that is adaptive to π's learning process.

Agent control. The problem of controlling an agent can be modeled as a Markov decision process (MDP), denoted by the tuple $\{\mathcal{S}, \mathcal{A}, p, r, \gamma\}$, where $\mathcal{S}$ and $\mathcal{A}$ represent the set of state space and action space respectively, p means the transition model between states, and r is the reward function. An agent of morphology $m_i \in M$ in state $s_t \in \mathcal{S}$ at time t takes action $a_t \in \mathcal{A}$ and the environment returns the agent's new state s_{t+1} according to the unknown transition function $p(s_{t+1}|s_t, a_t, m_i)$, along with associated reward $r_t = r(s_t, a_t)$. The goal is to learn the control policy $\pi^* : \mathcal{S} \to \mathcal{A}$ mapping states to actions that maximizes the expected return $\mathbb{E}[\mathcal{R}_t]$,

which takes into account reward at future time-steps $J(\pi) = \mathbb{E}[\mathcal{R}_t] = \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t r_t\right]$ with a discount factor $\gamma \in [0, 1]$, where H is the variable time horizon.

We use a graph neural network (GNN) for the control policy because (1) it can be generalized to different morphologies and learn fundamental control skills; (2) it captures the skeleton structure and encodes physical constraints. Specifically, we represent the agent's skeleton by a graph $G = (V, A, E)$ where each node $u \in V$ represents a joint and each edge $e \in E$ represents a bone connecting two joints. Each joint u is associated with a representation vector z_u storing attributes in A, e.g., bone length, motor strength, size. The GNN takes z_u as inputs and propagate messages across nodes on the graph to produce a hidden state for each node.

Morphology evolution. We model the morphology evolution as an MDP problem, denoted by the tuple $\{\mathcal{S}_m, \mathcal{B}, p_m, r_m, \rho\}$. $\mathcal{S}_m$ represents the set of state space, and a state $s^m_{\alpha t} \in \mathcal{S}_m$ at time-step αt is defined as $s^m_{\alpha t} = G_{\alpha t}$, where $G_{\alpha t}$ is the skeleton structure of agent. The action $b_{\alpha t}$ sampled from the set of action space $\mathcal{B}$ can modify the topology of the agent, that has three choices in $\{AddJoint, DelJoint, NoChange\}$. The transition dynamics $p_m(s^m_{\alpha t+1}|s^m_{\alpha t}, b_{\alpha t})$ reflects the changes in state and action. We define the reward function for π_m as the average improvement of training π with the current evolved morphology in different environments. The reward r_m at time-step αt is denoted as

$$r^m_{\alpha t} = \frac{1}{|E|} \sum_{\theta^E \in E} \left(\mathcal{R}(H, \theta^E, G_{\alpha t}) - \mathcal{R}(H, \theta^E, G_{\alpha t-1})\right) - \lambda \|b^m_{\alpha t}\|_2^2, \tag{5.1}$$

note that the first term in Eq. 5.1 indicates the average performance of the current morphology in diverse environments, and the second term is a cost for each action taken by π_m that modifies the morphology of the agent. E is the set of environments used for evaluation.

Environment evolution. We model environment evolution as an MDP problem, denoted by the tuple $\{\mathcal{S}_e, \mathcal{C}, p_e, r_e, \rho\}$. The state $s^e_{\beta t} = (G_{\beta t}, \theta^E_{\beta t})$ in the set of

state space $\mathcal{S}_e$ includes the agent's topology $G_{\beta t} = (\mathcal{V}_{\beta t}, \mathcal{E}_{\beta t})$ and the parameter of environment $\theta_E \in \Theta_E$. The transition dynamics $p_e(s^e_{\beta t+1}|s^e_{\beta t}, c_{\beta t})$ reflects the changes in the state by taking an action at time-step βt. The action $c_{\beta t}$ will directly change the environment parameters. We define the reward for π_e as the learning progress of multiple morphologies in the current environment. Thus, training π_e will encourage π_m to optimize the morphology of better generalization sooner. The reward r_e at time-step βt is denoted as

$$r^E_{\beta t} = \mathcal{P}_{\beta t} - \mathcal{P}_{\beta t-1}, \tag{5.2}$$
$$where, \quad \mathcal{P}_{\beta t} = \mathcal{R}(H, \theta^E_{\beta t}, G_{\beta t}) - \mathcal{R}(H, \theta^E_{\beta t-1}, G_{\beta t}).$$

in Eq. 5.2, $\mathcal{P}_k$ is the learning progress of the RL agent in an environment defined on $\mathcal{R}(H, \theta^E_{\beta t}, G_{\beta t})$, which denotes the expected return of the RL agent $G_{\beta t}$ of H time-steps evaluated in environment θ^E_k.

Short evaluation window. Since r_m and r_e are respectively based on the training process of π on different morphologies or environments, it is cost but necessary to rollout π periodically to collect data. However, Hejna et al. (2021) demonstrates that evaluating with short horizon is enough to provide informative feedback to reflect the training process of the current policy. In light of this, we run a short evaluation window for π after every period of environment steps taken by the agent. In each evaluation window, we rollout multiple morphologies of the best performance by π in several most recent environments, and then we can calculate r_m and r_e based on the evaluation results.

5.2 Algorithm of MECE

In MECE, we need jointly train three policies: control policy π learns to complete tasks, morphology policy π_m learns to evolve the agent's morphology to be more adaptive to diverse environments, and environment policy π_e learns to modify

the training environments more challenging. At first glance, training multiple RL policies may appear more difficult than training a single RL policy, as it needs the collection of additional data via interaction. However, MECE enables three policies to help in each other's training through the co-evolving mechanism, where each policy is trained on a curriculum of easy-to-hard tasks utilizing dense feedback from the training experience of other policies. By iterating this co-evolution process, MECE considerably improves the training efficiency of each policy, resulting in an RL agent of morphology that is more generalizable across environments.

Algorithm 7 MECE

1: **Initialize:** control policy π, morphology policy π_m, environment policy π_e, dataset $\mathcal{D} \leftarrow \emptyset$, initial agent morphology m_0, initial environment parameter θ_0^E;
2: **while** Not reaching max iterations **do**
3: **for** $t = 0, 1, \cdots, \tau_{\max}$ **do**
4: Sample control action $a_t \sim \pi$;
5: $s_{t+1} \leftarrow \mathcal{T}(s_{t+1}|s_t, a_t)$;
6: $r_t \leftarrow$ environment reward;
7: $m_{t+1} \leftarrow m_t$, $\theta_{t+1}^E \leftarrow \theta_t^E$;
8: Store $(s_t, r_t, a_t, s_{t+1}, m_t, \theta_t^E)$ into $\mathcal{D}$;
9: **end for**
10: Update π with PPO using samples in $\mathcal{D}$;
11: $(m_{t+1}, \theta_{t+1}^E) = \text{CO-EVO}(m_t, \pi, \theta_t^E)$;
12: **end while**

In each episode, the control policy π starts from training on an initial agent in a randomly sampled environment (refer to Line.3-9 in Alg. 7). This initial agent's skeleton structure is relatively simple and is easy to learn according to its naive

morphology. Even though the naive agent is not generalized due to its constrained morphology, it can initially provide dense feedback for training π_m and π_e. On the other hand, beginning from an agent with a random morphology is theoretically feasible but inefficient when initializing a complicated morphology that is difficult to learn to control and yields uninformative feedback for training the other policies.

We assume that the control policy is now mature on the current morphology after training. Then MECE proceeds to a series of co-evolution phases, where a phase is associated with achieving robustness across the evolved morphologies and environments (refer to Line.11 in Alg. 7). In each phase, as shown in Line.3-4 of Alg. 8, we first evaluate the control policy to collect rewards for training π_m and π_e. Note that this step is not cost, as the RL agent executes environment steps with a short horizon. After that, we alternately apply π_m and π_e to co-evolve the agent's morphology and the training environments based on two criteria of r_m and r_e (refer to Line.5 and 9 in Alg. 8).

Dynamic update window based on the reward criteria. As mentioned in Section 5.1, r_m and r_e indicate the learning progress of the current morphology in the training environment respectively. In particular, a nominal value of r_m corresponds to the adaptability of the current morphology to different environments and indicates that training with the current morphology cannot significantly increase π's performance. In this instance, we should apply π_m to optimize the morphology to increase its generalization to environments. Similarly, a minor r_e indicates that modifying the agent's morphology will have a negligible effect on its performance in the current environment, and the environment should be modified to be more challenging to boost the morphology evolution. As a result, we employ two independent criteria for r_m and r_e to formulate a dynamic update window, which allows MECE to adapt to the learning progress of morphology and environment and apply π_m or π_e to produce corresponding evolution and alterations.

Algorithm 8 CO-EVO

Input: Current morphology $m_{\alpha t}$, control policy π, training environment $\theta^E_{\beta t}$, dataset $\mathcal{D}$.
Procedure CO-EVO($m_{\alpha t}, \pi, \theta^E_{\beta t}$):
Evaluate π with morphology $m_{\alpha t}$ in $\theta^E_{\beta t}$;
Calculate reward r_m and r_e following Eq. 5.1 and Eq. 5.2;
if $r_m \leq \delta_m$ **then**
    Apply π_m to modify $m_{\alpha t}$ to $m_{\alpha t+1}$;
    Update π_m with PPO using samples in $\mathcal{D}$;
else
    if $r_e \leq \delta_e$ **then**
        Apply π_e to modify $\theta^E_{\beta t}$ to $\theta^E_{\beta t+1}$;
        Update π_e with PPO using samples in $\mathcal{D}$;
    end if
end if
Return $(m_{\alpha t+1}, \theta^E_{\beta t+1})$

We now have an agent of newly evolved morphology or a modified training environment, and then forward to the next iteration of training the control policy π on them. Using r_m and r_e, MECE has achieved the alternating evolution of morphology and environment. MECE not only improves the robustness of the morphology to various environment, but also improves the learning efficiency of the policies through the use of dense feedback.

5.3 Experiments

We design our experiments to answer the following questions: (1) Given dynamic environments, does our method outperform previous methods in terms of convergence speed and final performance? (2) Does our method create agents of better generalization to diverse environments?

5.3.1 Environments Setup

Diverse environments varying features that require the agent to evolve out different morphologies, e.g., a bipedal agent has too short legs to go across a high obstacle, or a agent of four legs navigate on rough road more smoothly than one of one/two legs. In our experiments, we conduct three environments that requires various morphologies to complete the tasks. We use a tuple θ^E of environment parameters to denote the possible physical features of environments.

In experiments, we have three types of environments: **(1) 2d locomotion:** In the 2D locomotion environment, agents are limited to movement in the X-Z plane. The state s_T is given by the final (x, z) positions of the morphology joints. We evaluate morphologies on three tasks: running forwards, running backwards, and jumping. Environment policy π_e takes actions to change the roughness of terrains, that is controlled by θ^E. **(2) 3d locomotion:** where a 3D agent's goal is to move as fast as possible along x-axis and is rewarded by its forward speed along x-axis.

Environments have a fixed number of obstacles and terrains of different roughness which are controlled by θ^E. π_e can not only change the roughness of terrains in the environments, also learns to modify the average distance between obstacles. **(3) Gap crosser:** Gap Crosser, where a 2D agent living in an $xz-$plane needs to cross periodic gaps and is rewarded by its forward speed. The width of the gap is controlled by π_e. In order to avoid unlearnable environments, the upper and lower limits of the gap width are limited.

Baselines. We compare our method with the following baselines that also optimize both the skeletal structure and joint attributes of an agent. (1)Neural Graph Evolution (NGE) (Wang et al., 2019d), which is an ES-based method that uses GNNs to enable weight sharing between an agent and its offspring. (2) Transform2Act (Yuan et al., 2022), that optimizes an RL policy to control the evolution of agents. Note that both baselines do not have diverse environments in the original code. For a fair comparison, we apply an environment set randomly sampled from a uniform distribution in the training phase for them. (3) enhanced POET (Wang et al., 2020), that pertains the modification of the training environments that are paired with agents possessing distinct yet unchanging morphologies. Given the absence of morphology optimization/evolution in enhanced POET, we conducted a random sampling of six sets of agents, each comprising five agents with distinct morphologies, resulting in a total of 30 agents. Subsequently, we selected the top-performing agent from each set for the test.

5.3.2 Implementation of Control Policy and Morphology Policy

we represent the agent's skeleton by a graph $G = (V, A, E)$, where each node $u \in V$ represents a joint and each edge $e \in E$ represents a bone connecting two joints. Each joint u is associated with a representation vector z_u storing attributes in A, e.g., bone length, motor strength, size. The GNN takes z_u as inputs and

propagate messages across nodes on the graph to produce a hidden state for each node. Both the control policy π and the morphology policy π_m take each node's hidden state as their inputs and output their actions.

Implementation of morphology poicy π_m MuJoCo agents are specified using XML strings, during the morphology modify phase, we represent each agent's skeleton structure as an XML string and modify its content based on the morphology actions. The action of π_m $\{DelJoint, AddJoint\}$ is applied to each node. Once the action is taken and the skeleton is changed, the graph architecture changes and thus the nodes' hidden states changes. More specifically, An agent is initialized with only one head joint and several (or zero) Lv1 body joints directly connected to the head joint. In each iteration, the morphology policy will traverse each joint (using GNN) to decide whether to add an Lv1 body joint connected to the head joint, or an Lv2 body joint connected to a Lv1 body joint (i.e., adding a new bone). Then, the morphology policy will further traverse each joint to modify its attributes (bone length, motor strength, size) on the morphology.

Modelling the morphology evolution as an MDP. In MECE, we aim to train an agent π_m that can design the morphology and modify it when adapting to new environments. And training π_m requires interactions with the MDP to get reward of each modification. This is more efficient than conventional morphology evolution methods that randomly search for a better morphology because the policy learns to explore the space more efficiently through (1) interactions with MDP using different morphologies; and (2) structural constraint and correlation captured by the GNN.

5.3.3 Implementation of Environment Policy with Environments

In this part, we share further information about the three experiment environments. A simple skeleton structure is used to initialize the agent. Each joint of

the agent is connected to its parent joint via a hinge. The environment state of each joint contains its joint angle and velocity. In addition to the height, phase, and global velocity, we also include additional information for the root joint, such as the phase and height. Zero padding is employed to ensure that the length of each joint state is same. The attribute characteristic of each joint includes the bone vector, bone size, and motor gear value. Using a loosely-defined attribute range, each attribute is normalized within the interval $[-1, 1]$.

2d locomotion The agent in this environment lives inside an xz-plane with a terrain ground. Each joint of the agent is allowed to have a maximum of three child joints. For the root joint, we add its height and 2D world velocity to the environment state. The enviornment reward function is defined as:

$$r_t = |p^x_{t+1} - p^x_t|/\delta t + 1, \tag{5.3}$$

where p^x_t denotes the x-position of the agent and $\delta t = 0.008$ is the time step. An alive bonus of 1 is also used inside the reward. The episode is terminated when the root height is below 1.4.

3d locomotion The agent in this environment lives freely in a 3D world with an uneven ground. There are a fixed number of obstacles randomly scattered on the surface. Each joint of the agent is allowed to have a maximum of three child joints except for the root. For the root joint, we add its height and 3D world velocity to the environment state. The reward function is defined as:

$$r_t = |p^x_{t+1} - p^x_t|/\delta t - \omega \cdot \frac{1}{J} \sum_{u \in V_t} \|a_{u,t}\|^2, \tag{5.4}$$

where $\omega = 0.0001$ is a weighting factor for the control penalty term. u denotes the each node of the skeleton structure V_t of the agent at time-step t. J is the total number of agent's joints, and the time step $\delta t = 0.04$.

Gap crosser The agent in this environment lives inside an xz-plane. The terrain of this environment includes a periodic gap. The height of the initial terrain is at 0.2. The agent must cross these gaps to move forward. Each joint of the agent is allowed to have a maximum of three child joints. For the root joint, we add its height, 2D world velocity, and a phase variable encoding the periodic x-position of the agent to the environment state. The reward function is defined as:

$$r_t = |p^x_{t+1} - p^x_t|/\delta t + 0.1, \tag{5.5}$$

where the time step $\delta t = 0.08$. An alive bonus of 0.1 is used inside the reward. The episode is terminated with the root height is below 1.5.

Implementation of environment policy π_e Environment policy modify the training environment by taking actions to change the environment parameters. For locomotion environments, we sample terrain height maps using random Gaussian mixtures. There is an environment tuple of two parameters to control the generation of terrains. The first environment parameter is the maximum height of the terrain, which is limited to 2.4(2.7) in 2d (3d) locomotion. The second environment parameter is to control the variance of the sampled environments, which is restricted in $[2.4, 7.2]$ for 2d, and $[2.7, 5.4]$ for 3d. In 3d locomotion, we have one more environment parameter in the tuple to control the averaging spacing between the obstacles, which is restricted in $[1.6, 4.4]$.

5.3.4 Hyperparameters in MECE

In this section, we present the hyperparameters searched and used for MECE in Tab. 5.1 and the hyperparameters for baseline in Tab. 5.2 and Tab. 5.3. All models are implements by PyTorch (Paszke et al., 2019). For control policy and morphology policy, we use the PyTorch Geometric package (Fey and Lenssen, 2019) and GraphConv (Morris et al., 2019) as the GNN layer. When training the policy

with PPO, we adopt generalized advantage estimation (GAE) (Schulman et al., 2016).

For fair comparison, we employ the same GNN architecture for MECE and baselines. In addition, we ensure that the number of simulation steps used for optimization is the same for both the baseline methods and ours. MECE and Transform2Act optimizes the policy with a batch size of 50000 for 1000 epochs, totaling 50 million simulation steps. NGE employs a population of 20 agents, and each agent is trained with a batch size of 20,000 for 125 generations, totaling 50 million simulation steps.

5.3.5 Main results

In Fig. 5.1(a)-(c), we report the average performance of MECE and all baseline methods in test environments for three settings. The accumulated rewards are averaged over 12 UNSEEN and randomly sampled environments and 6 different random seeds. In terms of the learning efficiency and final performance, MECE outperforms baseline methods in all environments. The results show that while Transform2Act's final performance can be improved by periodically changing the training environments, MECE's learning efficiency is still notably higher than that of Transform2Act. The best morphologies found by each approach are listed in Fig. 5.1(d)-(f) for easier comparison. For 2d locomotion MECE develops a morphology that resembles a crawling animal that can move fast over terrain. In 3D locomotion, MECE has developed a spider-like framework. MECE evolves a more streamlined structure in comparison to Transform2Act so that it can more possibly avoid obstacles. Finally, the Hopper-like agents developed by Transform2Act and MECE are comparable in the Gap crosser that can jump across gaps. Overall, MECE-optimized morphologies have superior structural symmetry and are more consistent with the characteristics of biological evolution in various environments.

5.3.6 Ablation Study and Analysis

We product out a series of ablation studies to prove the effectiveness of each part in MECE. We list the introduction information of each ablation study in Tab. 5.4, and report the results in Fig. 5.2. Note that in each ablation study, we only change one part of MECE and maintain the other parts the same as the original MECE, and all results are the test performance evaluated on the same environment set in Fig. 5.1 (b).

Ablation Study I. In this ablation study, we focus on the effectiveness of π_e for the training and report the results in Fig. 5.1(a). Note that the initial environment is relatively simple, while the final environment that evolved by π_m is more challenging. The results show that the evolved morphology and control policy trained in the diverse environments are more general than trained in the fixed environment, no matter the environment is simple or not. On the other hand, compare the performance of MECE with MECE (periodic envs), we can find that π_e helps the training in terms of the efficiency and the final performance. This is because training in the environment modified by π_e, compared to the randomly generated one, avoids the learning gap between two extremely different environments, and is smoother for π and π_m to learn. On the other hand, π_e learns to generate environment to accelerate the learning progress of π and π_m, which is shown clearly in the learning curves of 10 - 20 million simulation steps in Fig. 5.2(a).

Ablation Study II. The purpose of this ablation study is to demonstrate how π_m can produce morphology that is more adaptable to diverse environments and increase learning effectiveness. The results are shown in Fig. 5.2(b). When comparing the learning curves for MECE (random morph) and MECE (original), the former has a far higher early learning efficiency than the latter. This is due to π_e remaining constant and the environment not changing significantly, which pre-

vents the morphology from being adequately adapted to a variety of habitats, even some comparable environments. Theoretically, the "fixed morph-final" morphology should be able to be more adaptive to environments and, with training, reach competitive performance. Although it performs better than "random morph" and "fixed morph-initial," "MECE(original)" is still far behind. The findings indicate that training an agent with a complex morphology (i.e., MECE(fixed morph-final)) in diverse environments is difficult and ineffective. Therefore, it is imperative to implement an automated curriculum that encompasses diverse environments and morphologies in MECE to effectively train an adaptable agent.

Ablation Study III. Through a co-evolutionary process of adaptive criterion on r_m and r_e, π_m and π_e learn to evolve the morphology and environment. To determine whether this design is effective, we perform an ablation experiment. The results are shown in Fig 5.2(c). For "MECE (fixed evaluation window)", every fixed number of environment steps, π_m and π_e take actions to change the agent's morphology and environment. The results indicate that by removing the dynamic window, MECE's learning effectiveness has drastically decreased. It is challenging to coordinate the training developments of the three policies, particularly the control policy, without a dynamic evaluation window. Only a somewhat steady control policy, as was already mentioned, can give the other two reliable feedback for training. Moreover, the performance of Transform2Act is taken into account for easy comparing. Even though "MECE(fixed evaluation window)" is less effective, a competitive final performance to Transform2Act is still feasible.

Ablation Study IV. We designed this ablation investigation to verify the reward of π_m and π_e in MECE, and the findings are displayed in Fig. 5.2(d). Note that just one incentive is altered in each scenario in this experiment; all other rewards remain unchanged. We explore two scenarios for π_m: the first involves removing the reward ("MECE (reward-i)"), same to the setting of transform2Act), and the sec-

ond involves substituting the formal of r_e ("MECE (reward-ii)"), i.e., accelerating learning progress. We take into account employing learning progress (in the form of r_m, "MECE (reward-iii)") for π_e. The results show that the original design has obvious advantages in both learning efficiency and final performance. For the issues addressed by MECE, π_m can be thought of as the meta learner of π, whereas π_e is the meta learner of the former. In order to help π perform better on various tasks, π_m learns to modify the agent's morphology during training, and π_e speeds up both of their learning processes. Therefore, in both theoretical and practical tests, the original design is the most logical.

5.3.7 Case study: Co-evolution between morphology and environment

To further comprehend the co-evolution of morphology and environment in MECE, we perform a case study in 2d locomotion and present results in Fig. 5.3. The y-axis of the point plot indicates the training environment's roughness. Fig. 5.3(a)-(f) are schematic descriptions of the current morphology and the training environment corresponding to it.

Generally, an effective training environment should meet two conditions simultaneously. The first is learnable, and the RL agent can collect training data via exploration. The second factor is the level of difficulty; environments that are either too easy or too challenging may reduce the learning efficiency of the RL agent. Therefore, in 2D locomotion, a reasonable environment is one in which the environment's roughness is quite large (challenging), but not so large as to render the agent unconquerable (learnable) during exploration. Using the results of the point-plot, the environment policy can ensure that the ratio of environmental unevenness during the training process to the height of the current agent is approximately 1, which satisfies the training environment criteria. At contrast, despite the fact that the randomly generated environment can ensure the ideal in some phases, it is ex-

tremely unstable (the standard deviation is visibly high), which would undoubtedly lower the learning effectiveness.

Specifically, comparing Fig. 5.3(a) and (c), the environment generated by π_e in the early stage of training is unlearnable for the current agent, whereas in the middle stage of training, the training environment is relatively flat, but challenging, especially in terms of the number of occurrences of a particular feature. The roughness of the environment on the right fluctuates frequently. Note again compared to Fig. 5.3(d) that the right side of the randomly produced environment is unlearnable for the current agent in the middle of training. This phenomena does not occur in MECE because π_e modifies the environment to be morphology-adaptive. Comparing Figs. 5.3(e) and (f), the environment formed by π_e's slope exhibits noticeable but relatively subtle alterations. This is due to the fact that the existing morphology has a high degree of adaptation to varied contexts, and environments with more extreme alterations have less effect on the morphology's optimization. To better train the control policy, an environment of moderate difficulty should be adopted.

5.4 Discussion

In this section, we offer a framework for morphology-environment co-evolution that evolves the agent's morphology and the training environment in alternation. Compared to previous morphology optimization approaches, ours is more sample-efficient and learns to develop morphologies that are more robust across diverse environments. Experiments demonstrate that our approach outperforms baseline methods in terms of convergence speed and overall performance. In the future, we are interested in using curriculum RL approaches to further improve the learning efficiency of our approach so that it can adapt to more challenging environments and tasks.

Hyperparameter	Values Searched & Selected
Policy GNN Layer Type	GraphConv
Number of morphologies in SEW	$2, \mathbf{3}, 4$
Number of environments in SEW	$2, 3, \mathbf{4}, 5$
GNN Size (π_m)	$(32, 32, 32), (64, 64, 64), (128, 128, 128), (\mathbf{256}, \mathbf{256}, \mathbf{256})$
GNN Size (π)	$(32, 32, 32), (\mathbf{64}, \mathbf{64}, \mathbf{64}), (128, 128, 128), (256, 256, 256)$
Hidden layers for π_e network	$\mathbf{2}, 3, 4$
Hidden units for π_e network	$\mathbf{200}, 300, 400$
Policy Learning Rate (π)	$\mathbf{5e-5}, 1e-4, 3e-4$
Policy Learning Rate (π_m)	$\mathbf{5e-5}, 1e-4, 3e-4$
Policy Learning Rate (π_e)	$\mathbf{3e-4}, 5e-4, 1e-4$
Value GNN Layer Type	**GraphConv**
Value Activation Function	Tanh
Value GNN Size	$(\mathbf{64}, \mathbf{64}, \mathbf{64}), (128, 128, 128), (256, 256, 256)$
Value MLP Size	$(256, 256), (\mathbf{512}, \mathbf{256}), (256, 256, 256)$
Value Learning Rate	$1e-4, \mathbf{3e-4}$
PPO clip ϵ Pize	$\mathbf{0.2}$
PPO Batch Size	$10000, 20000, \mathbf{50000}$
PPO Minibatch Size	$512, \mathbf{2048}$
Num. of PPO Iterations Per Batch	$1, 5, \mathbf{10}$
Num. of Training Epochs	**1000**
Discount factor γ	$\mathbf{0.95}$
GAE λ	$\mathbf{0.99}, 0.995, 0.997, 0.999$

Table 5.1 : Hyperparameters searched and used by MECE. The bold numbers among multiple values are the final selected ones.

Hyperparameter	Values Searched & Selected
Num. of Skeleton Transforms N_s	$3, \mathbf{5}, 10$
Num. of Attribute Transforms N_z	$\mathbf{1}, 3, 5$
Policy GNN Layer Type	GraphConv
JSMLP Activation Function	Tanh
GNN Size (Skeleton Transform)	$(64, 64, 64), (\mathbf{128}, \mathbf{128}, \mathbf{128}), (256, 256, 256)$
JSMLP Size (Skeleton Transform)	$(\mathbf{256}, \mathbf{256}), (512, 256), (256, 256, 256)$
GNN Size (Attribute Transform)	$(64, 64, 64), (\mathbf{128}, \mathbf{128}, \mathbf{128}), (256, 256, 256)$
JSMLP Size (Attribute Transform)	$(\mathbf{256}, \mathbf{256}), (512, 256), (256, 256, 256)$
GNN Size (Execution)	$(\mathbf{64}, \mathbf{64}, \mathbf{64}), (128, 128, 128), (256, 256, 256)$
JSMLP Size (Execution)	$(\mathbf{128}, \mathbf{128}), (256, 256), (512, 256), (256, 256, 256)$
Diagonal Values of Σ^z	$1.0, 0.04, \mathbf{0.01}$
Diagonal Values of Σ^e	$\mathbf{1.0}, 0.04, 0.01$
Policy Learning Rate	$\mathbf{5e-5}$
Value GNN Layer Type	**GraphConv**
Value Activation Function	Tanh
Value GNN Size	$(\mathbf{64}, \mathbf{64}, \mathbf{64}), (128, 128, 128), (256, 256, 256)$
Value MLP Size	$(256, 256), (\mathbf{512}, \mathbf{256}), (256, 256, 256)$
Value Learning Rate	$1e-4, \mathbf{3e-4}$
PPO clip ϵ Pize	$\mathbf{0.2}$
PPO Batch Size	**50000**
PPO Minibatch Size	**2048**
Num. of PPO Iterations Per Batch	$1, 5, \mathbf{10}$
Num. of Training Epochs	**1000**
Discount factor γ	$\mathbf{0.95}$
GAE λ	$\mathbf{0.995}$

Table 5.2 : Hyperparameters searched and used by Transform2Act in our version. The bold numbers among multiple values are the final selected ones.

Hyperparameter	Values Searched & Selected
Num. of Generations	**125**
Agent Population Size	$10, \mathbf{20}, 50, 100$
Elimination Rate	$\mathbf{0.15}, 0.2, 0.3, 0.4$
GNN Layer Type	GraphConv
MLP Activation	Tanh
Policy GNN Size	$(32, 32, 32), (\mathbf{64}, \mathbf{64}, \mathbf{64}), (128, 128, 128)$
Policy MLP Size	$(\mathbf{128}, \mathbf{128}), (256, 256), (512, 256)$
Policy Log Standard Deviation	$\mathbf{0.0}, -1.6$
Policy Learning Rate	$\mathbf{5e-5}$
Value GNN Size	$(32, 32, 32), (\mathbf{64}, \mathbf{64}, \mathbf{64}), (128, 128, 128)$
Value MLP Size	$(128, 128), (256, 256), (\mathbf{512}, \mathbf{256})$
Value Learning Rate	$\mathbf{3}\text{e}-\mathbf{4}$
PPO clip ϵ	**0.2**
PPO Batch Size	$\mathbf{20000}, 50000$
PPO Minibatch Size	**2048**
Num. of PPO Iterations Per Batch	**10**
Discount factor γ	$0.99, \mathbf{0.995}$
GAE λ	**0.95**

Table 5.3 : Hyperparameters searched and used by NGE in our version. The bold numbers among multiple values are the final selected ones.

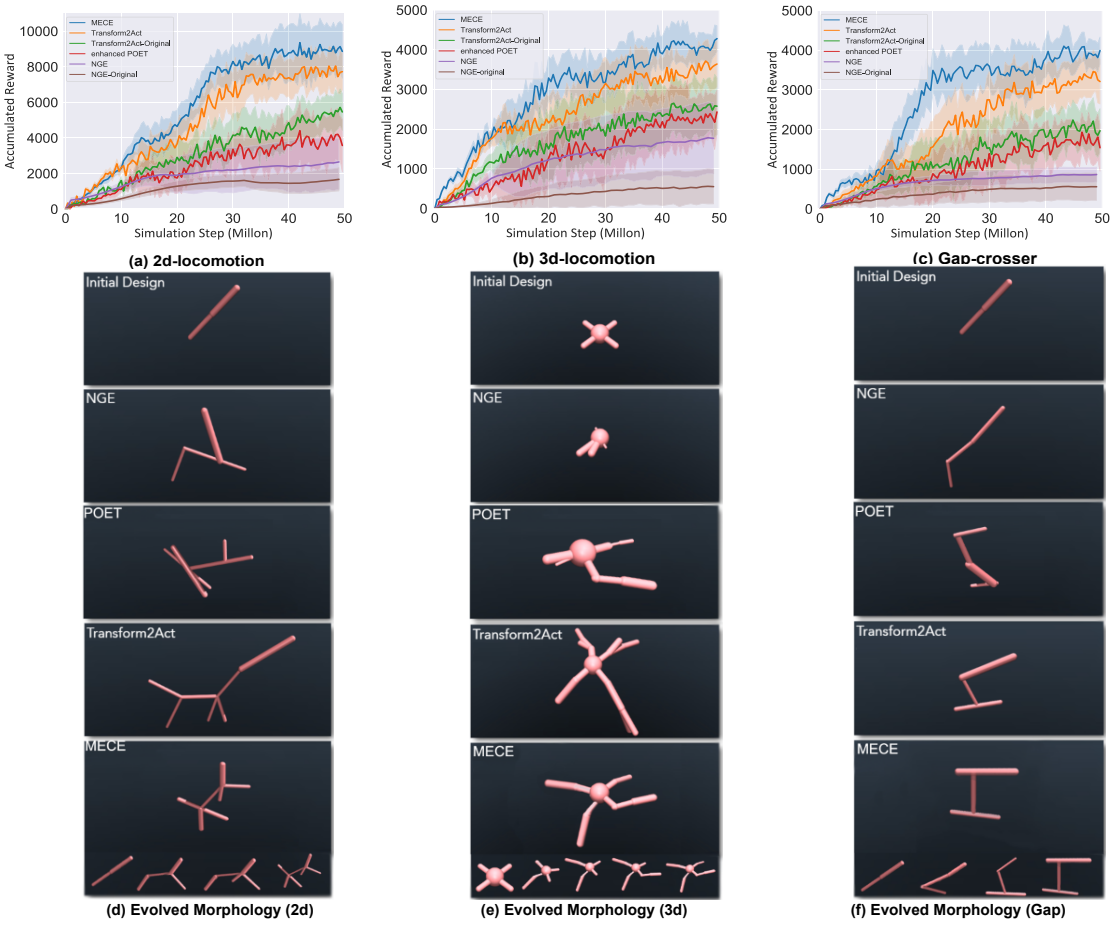

Figure 5.1 : **Baselines comparison and optimized morphology.** In Fig.(a)-(c), we report the evaluation results of MECE and baseline methods in three environments, and we plot the accumulated rewards (mean± std averaged over 6 random seeds) against the number of simulation steps for all methods. For fair comparison, we add periodically changing environments that randomly sampled from a fixed distribution in the training of baseline methods. MECE has better learning efficiency and final performance than all baseline methods. In Fig.(d)-(f), we list the optimal morphology that evolved by each method. Intuitively, the structural symmetry and environmental adaptability of the MECE-optimized morphology is better. Especially in 3d locomotion, the agent developed by MECE is better at navigating terrain and avoiding obstacles.

Ablation Study	Definition of Legends
Ablation study-I for π_e. Fig. 5.2(a)	**original**: Periodic environments generated by π_e. **periodic envs-random**: Periodic environments of randomly sampled. **fixed envs-initial**: Initial and easy environment. **fixed envs-final**: Fixed environment generated by π_e.
Ablation study-II for π_m. Fig. 5.2(b)	**original**: Dynamic morphology evolved by π_m. **random morph**: Randomly mutate the morphology. **fixed morph-initial**: Fixed agent of the initial morphology. **fixed morph-final**: Fixed agent of the final morphology evolved by π_m.
Ablation study-III for dynamic update window. Fig. 5.2(c)	**Original**: Dynamic evaluation window. **fixed update window**: Update π_e and π_m at a fixed frequency.
Ablation study-IV for r_m and r_e. Fig. 5.2(d)	**Original**: Follow the reward setting of π_m (π_e) in Eq. 5.1 (Eq. 5.2). **reward-i**: Remove the reward for π_m. **reward-ii**: change r_m to Eq. 5.1. **reward-iii**: change r_e to Eq. 5.2.

Table 5.4 : List of ablation studies. This list includes definitions for each set of controls as well as the objectives of ablation study I-IV. Since we only validated one design of MECE in each ablation study, only one design from each control group was different from MECE (original).

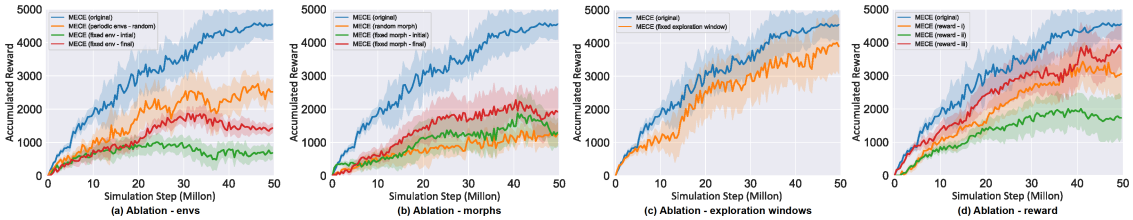

Figure 5.2 : **Ablation study.** We design four ablation studies to illustrate the advantages of each part in MECE. The results of ablation study I and II show the effectiveness of π_e and π_m on the learning efficiency and generalization. Ablation study III proves that applying π_e and π_m adaptive to the training process of the control policy can improve its robustness. In ablation study IV, we try different setting of the reward function of π_e and π_m, and the results prove that the original reward setting is optimal. More details of each ablation study can be found in Tab. 5.4.

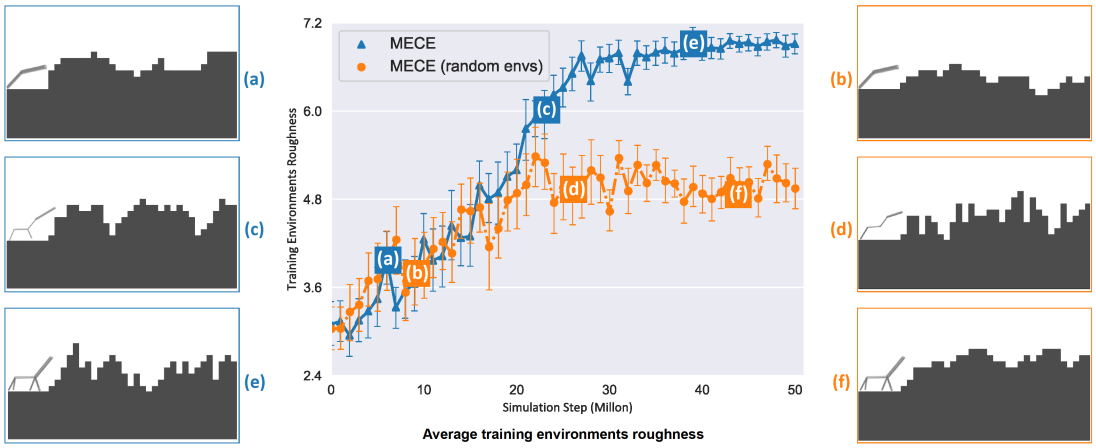

Figure 5.3 : **Case study results.** This figure more vividly illustrates a schematic diagram of the co-evolution of morphology and environment during training. The ordinate in the illustration shows the average environments' roughness changing as the training processes (6 random seeds). Although the training environments generated by π_e are similar to the randomly sampled at the beginning, it can be observed from the pointplot that π_e can guarantee more challenging and stable training environments. Fig.(a)-(f) compare the effectiveness of π_e that correspond to changes in the training environments (blue indicates original MECE, and orange indicates random environments). Contrarily, it is evident that no π_e is likely to result in exceptionally difficult environments (Fig.(d)), while the environment produced by π_e is challenging but learnable (Fig.(c)).

Chapter 6

Conclusions

The present study centers on investigating research that pertains to the acquisition of knowledge through a hybrid approach of reinforcement learning (RL)and curriculum learning. The Bellman equation-induced challenge of credit assignment in RL represents a constraint on the efficacy of this approach, which is further compounded by contextual limitations in its practical implementation. Although RL has exhibited significant potential in industrial and interdisciplinary research in recent years, it is limited by the following challenges. Conversely, the constraint is significantly intensified due to the sensitivity of RL towards its proximate milieu and its consequential reliance on it.

This thesis centers on three primary issues: the credit assignment problem, which arises from long-term and sparsely rewarded tasks; instability, which results from heightened environmental sensitivity; and generalization defects that prevent the RL agent from effectively binding with policy and environment. This thesis centers its attention on the following three concerns. The aforementioned concerns are effectively tackled and remedied via the implementation of the subsequent three curricular programs:

- CO-PILOT, a collaboratively training curriculum that integrates planning and reinforcement learning, and is designed around a sub-task curriculum that utilizes a tree structure. The objective of the course is to partition a task with a lengthy time frame into multiple tasks with shorter time frames, while concurrently utilizing planning. Propose sub-tasks as a means of providing

supplementary incentives to address the issue of insufficient rewards.

- EAT-C is an adaptation of CO-PILOT that integrates adjustments to the educational program and external perturbations to facilitate the reinforcement learning policy training process. This phenomenon results in a reduction of the sensitivity of RL policy to its surrounding environment and an increase in its ability to tolerate environmental disturbances within a specified range.

- MECE is centered around the reciprocal development of agent morphology and the training environment. The process allows for the evolution of morphology and control policy of reinforcement learning (RL) agents to suit diverse environments, thereby facilitating their adaptation to situations that exhibit substantial variations.

A diverse array of experiments and testing environments have been devised for each distinct class to authenticate their functionalities and aptitudes. The experimental results indicate that our ability to achieve performance on par with the state-of-the-art (SOTA) is accompanied by a noteworthy improvement in the learning efficacy of RL. Furthermore, a comprehensive ablation study was conducted to verify the rationale behind each individual component of the program. Ultimately, we provide a comprehensive analysis of each of our courses through the presentation of a case study, which serves to enhance the comprehension of their role within the broader framework of reinforcement learning training. In the future, it is possible to extend the auto-generated curriculum into real-world applications.

Bibliography

Amos, B., Rodriguez, I. D. J., Sacks, J., Boots, B. & Kolter, J. Z., 2018, 'Differentiable mpc for end-to-end planning and control', *NeurIPS*, .

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

Andrychowicz, M., Crow, D., Ray, A., Schneider, J., Fong, R. H., Welinder, P., McGrew, B., Tobin, J., Abbeel, P. & Zaremba, W., 2017b, 'Hindsight experience replay', .

Antos, A., Szepesvári, C. & Munos, R., 2008, 'Fitted q-iteration in continuous action-space mdps', vol. 20, pp. 9–16.

Asada, M., Noda, S., Tawaratsumida, S. & Hosoda, K., 2005, 'Purposive behavior acquisition for a real robot by vision-based reinforcement learning', *Machine Learning*, vol. 23, pp. 279–303.

Baykal, C. & Alterovitz, R., 2017, 'Asymptotically optimal design of piecewise cylindrical robots using motion planning', *Robotics: Science and Systems*, .

Bengio, Y., Louradour, J., Collobert, R. & Weston, J., 2009a, 'Curriculum learning', *International Conference on Machine Learning*, .

Bengio, Y., Louradour, J., Collobert, R. & Weston, J., 2009b, 'Curriculum learning', , p. 41–48.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. & Zaremba, W., 2016, 'Openai gym', .

Chen, B., Dai, B., Lin, Q., Ye, G., Liu, H. & Song, L., 2020a, 'Learning to plan in high dimensions via neural exploration-exploitation trees', .

Chen, T., He, Z. & Ciocarlie, M. T., 2020b, 'Hardware as policy: Mechanical and computational co-optimization using deep reinforcement learning', *CoRL*, .

Cheney, N., Bongard, J. C., SunSpiral, V. & Lipson, H., 2018, 'Scalable co-optimization of morphology and control in embodied machines', *Journal of The Royal Society Interface*, vol. 15.

Cheney, N., MacCurdy, R., Clune, J. & Lipson, H., 2013, 'Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding', *GECCO '13*, .

Chiang, H.-T. L., Faust, A., Fiser, M. & Francis, A., 2019, 'Learning navigation behaviors end-to-end with autorl', *IEEE Robotics and Automation Letters*, vol. 4, pp. 2007–2014.

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

da Silva, F. L. & Costa, A. H. R., 2018, 'Object-oriented curriculum generation for reinforcement learning', *Adaptive Agents and Multi-Agent Systems*, .

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

Dennis, M., Jaques, N., Vinitsky, E., Bayen, A., Russell, S. J., Critch, A. & Levine,

S., 2020, 'Emergent complexity and zero-shot transfer via unsupervised environment design', *ArXiv*, vol. abs/2012.02096.

Desai, R., Yuan, Y. & Coros, S., 2017, 'Computational abstractions for interactive design of robotic devices', *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1196–1203.

Elbanhawi, M. & Simic, M., 2014, 'Sampling-based robot motion planning: A review', *IEEE Access*, vol. 2, pp. 56–77.

Elbanhawi, M. & Simic, M., 2014, 'Sampling-based robot motion planning: A review', *IEEE Access*, vol. 2, pp. 56–77.

Elman, J. L., 1993, 'Learning and development in neural networks: the importance of starting small', *Cognition*, vol. 48, no. 1, pp. 71–99.

Exarchos, I., Jiang, Y., Yu, W. & Liu, C. K., 2021, 'Policy transfer via kinematic domain randomization and adaptation', *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 45–51.

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

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

Fang, M., Zhou, T., Du, Y., Han, L. & Zhang, Z., 2019, 'Curriculum-guided hindsight experience replay', *Advances in Neural Information Processing Systems*, , vol. 32.

Faust, A., Ramírez, O., Fiser, M., Oslund, K., Francis, A., Davidson, J. O. & Tapia, L., 2018, 'Prm-rl: Long-range robotic navigation tasks by combining reinforce-

ment learning and sampling-based planning', *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5113–5120.

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

Fey, M. & Lenssen, J. E., 2019, 'Fast graph representation learning with pytorch geometric', *ArXiv*, vol. abs/1903.02428.

Finn, C., Abbeel, P. & Levine, S., 2017, 'Model-agnostic meta-learning for fast adaptation of deep networks', *ArXiv*, vol. abs/1703.03400.

Florensa, C., Degrave, J., Heess, N., Springenberg, J. T. & Riedmiller, M. A., 2019, 'Self-supervised learning of image embedding for continuous control', *ArXiv*, vol. abs/1901.00943.

Florensa, C., Duan, Y. & Abbeel, P., n.d., 'Stochastic neural networks for hierarchical reinforcement learning.', *ICLR (Poster)*, .

Francis, A., Faust, A., Chiang, H.-T. L., Hsu, J., Kew, J. C., Fiser, M. & Lee, T.-W. E., 2020, 'Long-range indoor navigation with prm-rl', *IEEE Transactions on Robotics*, vol. 36, pp. 1115–1134.

Gupta, A., Fan, L. J., Ganguli, S. & Fei-Fei, L., 2022, 'Metamorph: Learning universal controllers with transformers', *ArXiv*, vol. abs/2203.11931.

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

Ha, S., Coros, S., Alspach, A., Kim, J. & Yamane, K., 2017, 'Joint optimization of robot design and motion parameters using the implicit function theorem', *Robotics: Science and Systems XIII*.

Haarnoja, T., Zhou, A., Abbeel, P. & Levine, S., 2018a, 'Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor', *ICML*, .

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P. & Levine, S., 2018b, 'Soft actor-critic algorithms and applications', *ArXiv*, vol. abs/1812.05905.

Hacohen, G. & Weinshall, D., 2019, 'On the power of curriculum learning in training deep networks', *International Conference on Machine Learning*, .

Hafner, D., Lillicrap, T. P., Fischer, I. S., Villegas, R., Ha, D. R., Lee, H. & Davidson, J., 2018, 'Learning latent dynamics for planning from pixels', *ArXiv*, vol. abs/1811.04551.

Hart, P. E., Nilsson, N. J. & Raphael, B., 1968, 'A formal basis for the heuristic determination of minimum cost paths', *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107.

Hejna, D. J., Abbeel, P. & Pinto, L., 2020, 'Hierarchically decoupled imitation for morphological transfer', *ArXiv*, vol. abs/2003.01709.

Hejna, D. J., Abbeel, P. & Pinto, L., 2021, 'Task-agnostic morphology evolution', *ArXiv*, vol. abs/2102.13100.

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

Howard, T. M. & Kelly, A., 2007, 'Optimal rough terrain trajectory generation for wheeled mobile robots', *The International Journal of Robotics Research*, vol. 26, pp. 141 – 166.

Hsu, D., Latombe, J. . & Motwani, R., 1997, 'Path planning in expansive configuration spaces', vol. 3, pp. 2719–2726 vol.3.

Ivanovic, B., Harrison, J., Sharma, A., Chen, M. & Pavone, M., 2018, 'Barc: Backward reachability curriculum for robotic reinforcement learning', *2019 International Conference on Robotics and Automation (ICRA)*, pp. 15–21.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C. & Kavukcuoglu, K., 2017, 'Population based training of neural networks', *ArXiv*, vol. abs/1711.09846.

Jiang, L., Zhou, Z., Leung, T., Li, L.-J. & Fei-Fei, L., 2017, 'Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels', *International Conference on Machine Learning*, .

Jiang, Y., Zhang, T., Ho, D., Bai, Y., Liu, C. K., Levine, S. & Tan, J., 2021, 'Simgan: Hybrid simulator identification for domain adaptation via adversarial reinforcement learning', *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2884–2890.

Jurgenson, T., Avner, O., Groshev, E. & Tamar, A., 2020a, 'Sub-goal trees - a framework for goal-based reinforcement learning', *ArXiv*, vol. abs/2002.12361.

Jurgenson, T., Avner, O., Groshev, E. & Tamar, A., 2020b, 'Sub-goal trees - a framework for goal-based reinforcement learning', *ArXiv*, vol. abs/2002.12361.

Kaelbling, L., 1993, 'Learning to achieve goals', *IJCAI*, .

Kaelbling, L. P. & Lozano-Pérez, T., 2011, 'Hierarchical task and motion planning in the now', *2011 IEEE International Conference on Robotics and Automation*, pp. 1470–1477.

Karaman, S. & Frazzoli, E., 2011, 'Sampling-based algorithms for optimal motion planning', *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894.

Karras, T., Aila, T., Laine, S. & Lehtinen, J., 2017, 'Progressive growing of gans for improved quality, stability, and variation', *ArXiv*, vol. abs/1710.10196.

Kavraki, L. E., Svestka, P., Latombe, J. . & Overmars, M. H., 1996, 'Probabilistic roadmaps for path planning in high-dimensional configuration spaces', *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580.

Kim, T.-H. & Choi, J., 2018, 'Screenernet: Learning self-paced curriculum for deep neural networks', *arXiv: Computer Vision and Pattern Recognition*.

Kingma, D. P. & Ba, J., 2015, 'Adam: A method for stochastic optimization', *CoRR*, vol. abs/1412.6980.

Kulkarni, T. D., Narasimhan, K., Saeedi, A. & Tenenbaum, J., 2016, 'Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation', *NIPS*, .

Kumar, M. P., Packer, B. & Koller, D., 2010, 'Self-paced learning for latent variable models', *NIPS*, .

Kurin, V., Igl, M., Rocktäschel, T., Boehmer, W. & Whiteson, S., 2021, 'My body is a cage: the role of morphology in graph-based incompatible control', *ArXiv*, vol. abs/2010.01856.

Lau, M. & Kuffner, J., 2005a, 'Behavior planning for character animation', *SCA '05*, .

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

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

Lavalle, S. M., 1998, 'Rapidly-exploring random trees: A new tool for path planning', .

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

Lee, L., Parisotto, E., Chaplot, D. S., Xing, E. & Salakhutdinov, R., 2018a, 'Gated path planning networks', *ICML*, .

Lee, S. Y., Choi, S.-I. & Chung, S.-Y., 2018b, 'Sample-efficient deep reinforcement learning via episodic backward update', *Neural Information Processing Systems*, .

Lee, Y. J. & Grauman, K., 2011, 'Learning the easy things first: Self-paced visual category discovery', *CVPR 2011*, pp. 1721–1728.

Levine, S., Lee, Y., Koltun, V. & Popovic, Z., 2011a, 'Space-time planning with parameterized locomotion controllers', *ACM Trans. Graph.*, vol. 30, pp. 23:1–23:11.

Levine, S., Lee, Y., Koltun, V. & Popović, Z., 2011b, 'Space-time planning with parameterized locomotion controllers', vol. 30, no. 3.

Liang, J., Hu, D. & Feng, J., 2020, 'Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation', *International Conference on Machine Learning*, .

Liao, T., Wang, G., Yang, B., Lee, R. P., Pister, K. S. J., Levine, S. & Calandra, R., 2019, 'Data-efficient learning of morphology and controller for a microrobot', *2019 International Conference on Robotics and Automation (ICRA)*, pp. 2488–2494.

Lillicrap, T., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. &

Wierstra, D., 2016, 'Continuous control with deep reinforcement learning', *CoRR*, vol. abs/1509.02971.

Liu, S., Lever, G., Merel, J., Tunyasuvunakool, S., Heess, N. M. O. & Graepel, T., 2019, 'Emergent coordination through competition', *ArXiv*, vol. abs/1902.07151.

Lowrey, K., Kolev, S., Dao, J., Rajeswaran, A. & Todorov, E., 2018, 'Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system', *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pp. 35–42.

Luck, K. S., Amor, H. B. & Calandra, R., 2019, 'Data-efficient co-adaptation of morphology and behaviour with deep reinforcement learning', *CoRL*, .

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

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D. & Kavukcuoglu, K., 2016, 'Asynchronous methods for deep reinforcement learning', *ICML*, .

Morerio, P., Cavazza, J., Volpi, R., Vidal, R. & Murino, V., 2017, 'Curriculum dropout', *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 3564–3572.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G. & Grohe, M., 2019, 'Weisfeiler and leman go neural: Higher-order graph neural networks', *ArXiv*, vol. abs/1810.02244.

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

Narvekar, S., Sinapov, J., Leonetti, M. & Stone, P., 2016, 'Source task creation for curriculum learning', *Adaptive Agents and Multi-Agent Systems*, .

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

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. & Chintala, S., 2019, 'Pytorch: An imperative style, high-performance deep learning library', *NeurIPS*, .

Pathak, D., Agrawal, P., Efros, A. A. & Darrell, T., 2017, 'Curiosity-driven exploration by self-supervised prediction', *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 488–489.

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

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

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

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

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

Racanière, S., Lampinen, A., Santoro, A., Reichert, D. P., Firoiu, V. & Lillicrap, T., 2019, 'Automated curricula through setter-solver interactions', *ArXiv*, vol. abs/1909.12892.

Rajeswaran, A., Kumar, V., Gupta, A., Schulman, J., Todorov, E. & Levine, S., 2017, 'Learning complex dexterous manipulation with deep reinforcement learning and demonstrations', *ArXiv*, vol. abs/1709.10087.

Rickert, M., Brock, O. & Knoll, A., 2008, 'Balancing exploration and exploitation in motion planning', *2008 IEEE International Conference on Robotics and Automation*, pp. 2812–2817.

Ross, S., Gordon, G. & Bagnell, J., 2011, 'A reduction of imitation learning and structured prediction to no-regret online learning', .

Russell, S. J. & Norvig, P., 2003, *Artificial Intelligence: A Modern Approach*, 2nd edn., Pearson Education.

Salimans, T., Ho, J., Chen, X. & Sutskever, I., 2017, 'Evolution strategies as a scalable alternative to reinforcement learning', *ArXiv*, vol. abs/1703.03864.

Savinov, N., Dosovitskiy, A. & Koltun, V., 2018, 'Semi-parametric topological memory for navigation', *ArXiv*, vol. abs/1803.00653.

Schaul, T., Horgan, D., Gregor, K. & Silver, D., 2015a, 'Universal value function approximators', , pp. 1312–1320.

Schaul, T., Quan, J., Antonoglou, I. & Silver, D., 2015b, 'Prioritized experience replay', *CoRR*, vol. abs/1511.05952.

Schmidhuber, J., 2011, 'Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem', *Frontiers in Psychology*, vol. 4.

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

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D. & Graepel, T., 2019, 'Mastering atari, go, chess and shogi by planning with a learned model', .

Schulman, J., Moritz, P., Levine, S., Jordan, M. I. & Abbeel, P., 2016, 'High-dimensional continuous control using generalized advantage estimation', *CoRR*, vol. abs/1506.02438.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O., 2017, 'Proximal policy optimization algorithms.', *CoRR*, vol. abs/1707.06347.

Shu, T., Xiong, C. & Socher, R., 2018, 'Hierarchical and interpretable skill acquisition in multi-task reinforcement learning', *ArXiv*, vol. abs/1712.07294.

Sims, K., 1994, 'Evolving 3d morphology and behavior by competition', *Artificial Life*, vol. 1, pp. 353–372.

Srinivas, A., Jabri, A., Abbeel, P., Levine, S. & Finn, C., 2018, 'Universal planning networks', *ArXiv*, vol. abs/1804.00645.

Srivastava, R. K., Steunebrink, B. R. & Schmidhuber, J., 2012, 'First experiments with powerplay', *Neural networks : the official journal of the International Neural Network Society*, vol. 41, pp. 130–6.

Sukhbaatar, S., Kostrikov, I., Szlam, A. D. & Fergus, R., 2017, 'Intrinsic motivation and automatic curricula via asymmetric self-play', *ArXiv*, vol. abs/1703.05407.

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

Sutton, R. S., McAllester, D., Singh, S. & Mansour, Y., 1999, 'Policy gradient methods for reinforcement learning with function approximation', NIPS'99, MIT Press, Cambridge, MA, USA, p. 1057–1063.

Tamar, A., Levine, S., Abbeel, P., Wu, Y. & Thomas, G., 2016, 'Value iteration networks', .

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., de Las Casas, D., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T. P. & Riedmiller, M. A., 2018, 'Deepmind control suite', .

Tesauro, G., 1995, 'Temporal difference learning and td-gammon', *Commun. ACM*, vol. 38, pp. 58–68.

Todorov, E., Erez, T. & Tassa, Y., 2012, 'Mujoco: A physics engine for model-based control', *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033.

Trabucco, B., Phielipp, M. & Berseth, G., 2022, 'Anymorph: Learning transferable polices by inferring agent morphology', *ArXiv*, vol. abs/2206.12279.

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

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R.,

Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T. P., Kavukcuoglu, K., Hassabis, D., Apps, C. & Silver, D., 2019, 'Grandmaster level in starcraft ii using multi-agent reinforcement learning', *Nature*, pp. 1–5.

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

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

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

Wang, R., Lehman, J., Rawal, A., Zhi, J., Li, Y., Clune, J. & Stanley, K. O., 2020, 'Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions', *International Conference on Machine Learning*, .

Wang, T., Zhou, Y., Fidler, S. & Ba, J., 2019d, 'Neural graph evolution: Towards efficient automatic robot design', *ArXiv*, vol. abs/1906.05370.

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

Wu, Y., Tucker, G. & Nachum, O., 2019, 'The laplacian in rl: Learning representations with efficient approximations', *ArXiv*, vol. abs/1810.04586.

Wulfmeier, M., Ondruska, P. & Posner, I., 2015, 'Maximum entropy deep inverse reinforcement learning', *arXiv: Learning*.

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

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

Yu, W., Liu, C. K. & Turk, G., 2019, 'Policy transfer with strategy optimization', *ArXiv*, vol. abs/1810.05751.

Yuan, Y., Song, Y., Luo, Z., Sun, W. & Kitani, K. M., 2022, 'Transform2act: Learning a transform-and-control policy for efficient agent design', *ArXiv*, vol. abs/2110.03659.

Zhang, D., Yang, L., Meng, D., Xu, D. & Han, J., 2017, 'Spftn: A self-paced fine-tuning network for segmenting objects in weakly labelled videos', *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5340–5348.

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

Zhang, M., Yu, Z.-Q., Wang, H., Qin, H., Zhao, W. & Liu, Y., 2019, 'Automatic digital modulation classification based on curriculum learning', *Applied Sciences*.

Zhang, T., Guo, S., Tan, T., Hu, X. & Chen, F., 2020, 'Generating adjacency-constrained subgoals in hierarchical reinforcement learning', *ArXiv*, vol. abs/2006.11485.

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

Zhao, A., Xu, J., Konakovic-Lukovic, M., Hughes, J., Spielberg, A., Rus, D. & Matusik, W., 2020, 'Robogrammar: graph grammar for terrain-optimized robot design', *ACM Trans. Graph.*, vol. 39, pp. 188:1–188:16.