UNIVERSITY OF CALIFORNIA SAN DIEGO

Towards Unsupervised Goal Discovery: Learning Plannable Representations with Probabilistic World Modeling

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science

by

Meng Song

Committee in charge:

Professor Manmohan Chandraker, Chair
Professor Mikhail Belkin
Professor Henrik Christensen
Professor Sanjoy Dasgupta
Professor Truong Nguyen

2024

The Dissertation of Meng Song is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2024

DEDICATION

To my parents, for their unconditional love and support.

EPIGRAPH

The wound is the place where the Light enters you.
Let the beauty of what you love be what you do.

Rumi

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to my advisor for supporting my research throughout my PhD studies.

I would also like to thank Professor Sanjoy Dasgupta for unveiling the mathematical beauty and elegance of machine learning in his class and for teaching me mathematical thinking. My experience as a TA with him has profoundly transformed the way I study AI, shifting it from an engineering approach to a scientific one.

I am grateful to the music, poetry, art, sunshine, great minds in history, and similar souls behind the screens. The inspiration and resonance from them transcend time and space, supporting me through the hopeless days and nights.

Finally, I owe my deepest gratitude to my parents for their unconditional love and their willingness to sacrifice everything in support of my dreams.

Chapter 1, in part, is a reprint of the material as it appears in "Learning to Rearrange with Physics-Inspired Risk Awareness", Meng Song, Yuhan Liu, Zhengqin Li, Manmohan Chandraker, Risk Aware Decision Making Workshop at The Robotics: Science and Systems, 2022. The dissertation author was the primary investigator and author of this paper.

Chapter 2, in full, is a reprint of the material as it appears in "A Minimalist Prompt for Zero-Shot Policy Learning", Meng Song, Xuezhi Wang, Tanay Biradar, Yao Qin, Manmohan Chandraker, Task Specification Workshop at The Robotics: Science and Systems, 2024. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in part, is a reprint of the material as it appears in "Probabilistic World Modeling with Asymmetric Distance Measure", Meng Song, Geometry-grounded Representation Learning and Generative Modeling Workshop at International Conference on Machine Learning, 2024. The dissertation author was the primary investigator and author of this paper.

VITA

2009	Bachelor of Science in Software Engineering, Nankai University, China
2012	Master of Science in Pattern Recognition and Intelligent System, Nankai University, China
2014	Master of Science in Robotics, Carnegie Mellon University
2024	Doctor of Philosophy in Computer Science, University of California San Diego

PUBLICATIONS

Meng Song, "Probabilistic World Modeling with Asymmetric Distance Measure", *Geometry-grounded Representation Learning and Generative Modeling Workshop at International Conference on Machine Learning* (ICML) [Oral], 2024.

Meng Song, Xuezhi Wang, Tanay Biradar, Yao Qin, Manmohan Chandraker, "A Minimalist Prompt for Zero-Shot Policy Learning", *Task Specification Workshop at The Robotics: Science and Systems* (RSS), 2024.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, **Meng Song**, Eric P. Xing, Zhiting Hu, "RLPrompt: Optimizing Discrete Text Prompts with Reinforcement Learning", *Conference on Empirical Methods in Natural Language Processing* (EMNLP), 2022.

Meng Song, Yuhan Liu, Zhengqin Li, Manmohan Chandraker, "Learning to Rearrange with Physics-Inspired Risk Awareness", *Risk Aware Decision Making Workshop at The Robotics: Science and Systems* (RSS), 2022.

Zhengqin Li, Ting-Wei Yu, Shen Sang, Sarah Wang, **Meng Song**, Yuhan Liu, et al., "OpenRooms: An End-to-End Open Framework for Photorealistic Indoor Scene Datasets", *Conference on Computer Vision and Pattern Recognition* (CVPR) [Oral], 2021.

Yuzhe Qin, Rui Chen, Hao Zhu, **Meng Song**, Jing Xu, Hao Su, "S^4G: Amodal Single-view Single-Shot $\mathbb{SE}(3)$ Grasp Detection in Cluttered Scenes", *Conference on Robot Learning* (CoRL), 2019.

Meng Song, "One-shot Logo Detection in the Wild", *WiML Workshop at Conference on Neural Information Processing Systems* (NeurIPS), 2018.

Meng Song, Daniel Huber, "Automatic Recovery of Networks of Thin Structures", *International Conference on 3D Vision* (3DV) [Oral], 2015.

Meng Song, Fengchi Sun, and Karl Iagnemma, "Natural landmark extraction in cluttered forested environments", *IEEE International Conference on Robotics and Automation* (ICRA) [Oral], 2012.

Meng Song, Fengchi Sun, and Karl Iagnemma, "Natural feature based localization in forested environments", *IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS) [Oral], 2012.

ABSTRACT OF THE DISSERTATION

Towards Unsupervised Goal Discovery: Learning Plannable Representations with Probabilistic World Modeling

by

Meng Song

Doctor of Philosophy in Computer Science

University of California San Diego, 2024

Professor Manmohan Chandraker, Chair

Learning through interaction is a foundational principle in both human and animal learning. In a broad sense, intelligent agents can be formulated as goal-directed systems interacting with an uncertain environment. Despite the generality of this definition, a key challenge in computationally grounding it lies in how to effectively set up and represent goals and purposes. This dissertation explores this question through the lens of various machine learning paradigms.

We begin with the classical reinforcement learning formulation, which treats all goals as reward maximization. We demonstrate that, under a novel physics-inspired reward function,

the agent can learn to solve rearrangement tasks with an awareness of physical concepts such as mass and friction.

Next, we move to the imitation learning setting to lift the limitations of scalar rewards. Specifically, we study the role of different forms of prompts in communicating task specifications to a decision transformer. We show that in most robotics applications, providing a demonstration as a prompt during deployment is often impractical. Instead, using an essential task parameter along with a learnable prompt can achieve comparable or even better generalization in a zero-shot manner.

Finally, we shift our focus from imparting human knowledge about goals to enabling the agent to discover goals purely from data. We show that a geometric abstraction of probabilistic world dynamics can be embedded into the representation space through asymmetric contrastive learning. The resulting embedding space allows for irreversible transitions and facilitates subgoal discovery and planning.

Introduction

This dissertation studies the learning problem of a goal-directed agent interacting with the world. The agent's goal or task objective can be formalized within different machine learning paradigms, either as explicit supervision or as hidden structures. Our exploration begins with human-designed goals and progresses towards data-driven goals, unleashing the potential of unsupervised learning in discovering intrinsic goals and skills.

We begin with the reinforcement learning (RL) formulation of the sequential decision-making problem, where the agent's goal is to maximize a cumulative scalar reward signal through trial and error. A typical formula for encoding goals or objectives into the reward function is to parameterize it with the desired and undesired outcomes in accomplishing the task, as well as the associated costs. Maximizing the reward function then becomes equivalent to finding an optimal solution to the specified task. As an example, we develop a novel physics-inspired reward function and demonstrate how it enables the agent to solve an indoor rearrangement task while also discerning different masses and friction coefficients (Chapter 1).

Although the reward hypothesis [1] states that, "All of what we mean by goals and purposes can be well thought of in terms of reward maximization", realizing this requires translating goals and purposes into a preference relation on possible outcomes, which is often either incommensurable or involves crafting a complex reward function in practice. To avoid reward engineering, we consider communicating tasks to the model in an imitation learning setting, where the task specification is provided as an additional input that the model conditions on. In transformer-based models, this is known as a prompt—an expressive and flexible interface for task specification. Prompts can take various forms and modalities, such as demonstrations,

language instructions, coordinates, or visual goals. In Chapter 2, we examine the role and necessity of demonstrations as prompts for a decision-transformer and investigate what constitutes a minimal sufficient prompt that can achieve the same level of generalization.

While learning tasks defined by human knowledge provides the agent with basic capabilities, it does not represent the highest form of intelligence. In an unknown environment, a truly autonomous agent should be capable of setting its own goals based on its understanding of the environment and discovering various skills to achieve them. This raises a natural question: if an agent can freely explore an environment and learn to solve any tasks, what subgoals is it most likely to set? We frame this question as uncovering structures from data sampled from a Markov chain, which can be viewed as a form of unsupervised learning for temporal data. Unlike its counterpart for classification and reconstruction, the learned representation space must be plannable. More precisely, it should be equipped with a distance function that reflects the original state transitions. In Chapter 3, we show that contrastive learning can be adapted to learn such an asymmetric distance function in the embedding space, allowing for the representation of irreversible transitions. Moreover, it captures a geometric abstraction of the original Markov transition graph. Exploring this geometric interpretation leads to a method for identifying intrinsic bottleneck states as subgoals, and paves the way for improved planning algorithms.

Chapter 1

Learning Physics-Aware Rearrangement in Indoor Scenes

1.1 Introduction

There has been significant recent interest in agents that learn to execute a task by interacting with an environment, rather than passive perception [2, 3, 4, 5, 6, 7, 8, 9]. Understanding physical properties and accounting for them in deriving rewards can form a meaningful basis for exploring and interacting with the environment. While prior studies focus on learning physical laws or object properties from observations [10, 11, 12, 13, 14, 15], we consider an agent's ability to account for physical properties in a task-driven sequential decision process. Specifically, we study how physical notions of mass and friction coefficient affect policy learning in a reinforcement learning framework for rearrangement problems in indoor scenes.

We argue that awareness of physical properties such as mass and friction is fundamental for robot learning to accomplish real-world tasks. Robust navigation demands that the agent adapts to terrains with varying friction coefficients. When displacing an object, the agent must consider its mass to determine the optimal interaction strategy, including the appropriate pose, applied force, and associated costs. However, most existing robot learning tasks and objectives often overlook these factors. Typically, navigation and manipulation tasks are framed as a single goal state matching problem [16, 4, 5, 17, 18], where the goal state is fully specified by perception data (e.g. images [18], point clouds [5], language descriptions [16] or geometric

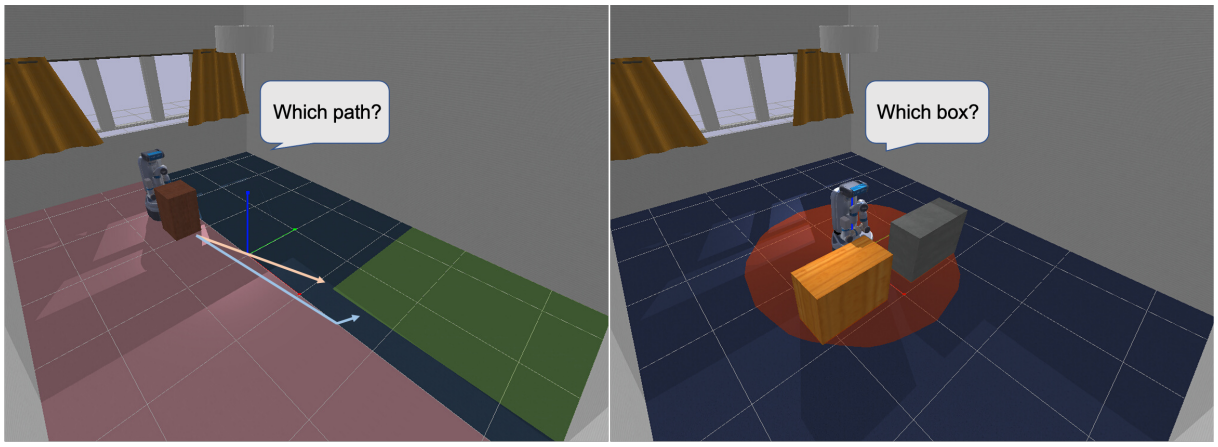


Figure 1.1. Physics-aware indoor rearrangement tasks. Left: The agent must learn to account for differing friction coefficients for the pink and blue sections of the floor, in order to efficiently push the box to the goal region (indicated in green). Right: The agent must learn to account for the difference in masses for the orange and gray boxes, and to expend the least physical cost in pushing one of them outside the circular region.

configurations [4, 17]). Aside from obstacle avoidance, most task objectives typically overlook how physical factors impact the mechanical process of reaching the goal state. As a result, they fail to equip the agent with a sufficient understanding of the environment's physical properties. In other words, an agent that only comprehends geometric concepts like shapes and distances can still accomplish these tasks without grasping essential physical notions, such as material properties or object masses.

To address these limitations, we introduce two novel and challenging indoor rearrangement tasks that involve a wheeled robot pushing boxes to designated goal regions (Fig. 1.1). In contrast to typical pushing tasks defined in terms of a single goal state, we allow a set of valid goal states with different physical properties. Finding the optimal task solution requires the agent's policy to reflect its understanding of friction and mass. In the *variable friction* pushing task, the agent is asked plan a trajectory to push the box to the target region across a floor with two different friction coefficients. In the *variable mass* pushing task, the agent is given two boxes with different masses and must push one of them outside the circle area.

To solve these tasks with an awareness of physics, the agent must minimize its cumulative physical cost in addition to accomplishing the task. We formulate this objective as a novel physics-

inspired reward function, which compels the agent to account for realistic physical interactions, such as the moment of the pushing force that a simplified point mass model cannot capture. Our experiments demonstrate that the learned policies under this new reward function align well with human intuition regarding mass and friction.

Besides evaluating the agent's awareness of physics, the proposed tasks referred to as pushing while moving *pushing while moving*, also unify the agent's navigation and object interaction skills. While a similar task design has been explored under the term interactive navigation, our approach differs by focusing on the agent's awareness of mass and friction variations, rather than solely enhancing its navigation abilities. Furthermore, our task design allows the agent to develop an understanding of mass and friction without requiring dexterous manipulation. The introduction of a physical efficiency metric also broadens the scope of environmental consciousness in robot intelligence, which current simulation environments [3, 4, 19, 2, 5, 20, 8] do not fully capture. Potential practical benefits of training such a physically efficient agent include running everyday household tasks in energy-saving modes and enabling robotic applications in energy-limited scenarios such as search-and-rescue.

1.2 Related Work

Physical Understanding

Physical reasoning and understanding have long been of interest in computer vision and machine learning communities [21, 10, 22, 23, 14, 24, 15, 25, 13, 26, 11, 12]. Modern learning-based methods usually formulate this problem as a future prediction problem. The underlying assumption is that if the model can accurately reconstruct and predict long-term observations, the learned representations must capture information about physical dynamics and properties. The physical knowledge can be modeled on different levels, ranging from latent variables indicating physical properties [25, 11] to a world model of object interactions [22, 23, 15]. However, a common limitation of these efforts is that the physical process is observed and learned in a passive way without the agent's continuous interventions driven by a task. Although the experimental

scenes evolve from simple ones such as objects sliding down ramps [25], falling down of a block tower [13] to complicated multi-object collisions, the intervention is only exerted at the very beginning (i.e. contextual bandit problem). This separation of visual understanding and actuation impedes the agent from verifying and improving its learned knowledge perpetually to accomplish real-world tasks.

Embodied AI

Recently, impressive progress in embodied AI research has been driven by applying reinforcement learning algorithms to visual navigation based tasks, ranging from navigating to a goal location [27], searching for objects from a specific category [28], transporting objects [7], to playing hide-and-seek game [29]. Most works emphasize tackling the long-horizon navigation challenges. This typically requires heuristics (e.g. geodesic distance to the goal), building maps or performing hierarchical planning in a large-scale indoor scene across multiple rooms [7, 3, 2, 4]. However, our tasks focus on rearrangement in a single room scene with no need for mapping or any distance information about the goals.

Object Rearrangement

A rich body of works have studied object rearrangement tasks [9, 3, 4, 19, 2, 5, 20, 8], spanning a variety of real-world applications such as table organization, storing groceries and house cleaning. However, most of these experimental testbeds have concentrated on tasks requiring robotic manipulators or magic pointer [3, 4], which either involve hand dynamics or are not physics-driven. Moreover, none of them have considered physical cost in their reward functions or evaluation metrics.

Material and Dynamics Randomization

While most recent robot learning studies are not focusing on the problem of understanding physical properties, some of them [2, 6] have incorporated randomization of object materials and dynamics properties (mass and friction) in the learning process. However, in these works, the aim of varying physical properties is to minimize their impact on generalization. The expectation

is that by sufficiently diversifying the training distribution of these variables, the algorithms will generalize effectively to unseen values, thereby improving sim-to-real transfer. This is orthogonal to the way we utilize the variation of physical properties. In our case, the physical causality is carefully encoded and highlighted in the task definition, and the agent is enforced to infer it by acting in the environment and maximizing the reward.

1.3 Problem Formulation

Reinforcement Learning

A reinforcement learning problem can be formalized as learning to control a dynamical system (i.e. environment) defined by a Markov decision process (MDP). An MDP $\mathscr{M}$ is a tuple $(\mathscr{S}, \mathscr{A}, P, r, \rho_0, \gamma)$, where $\mathscr{S}$ is the state space, $\mathscr{A}$ is the action space, ρ_0 defines the initial state distribution, $\gamma \in (0,1]$ is a discount factor and $r : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$ defines a task-specific reward function. P defines the dynamics of the environment, which can be written as a transition probability distribution $P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. The aim of reinforcement learning is to learn a policy π to maximize objective $J(\pi)$:

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{H} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{1.1}$$

where a finite-horizon trajectory τ is denoted by $\tau = (\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{s}_H, \mathbf{a}_H)$. Given MDP $\mathscr{M}$ and policy π, its distribution $p_\pi(\tau)$ can be derived as:

$$p_\pi(\tau) = \rho_0(\mathbf{s}_0) \prod_{t=0}^{H} \pi(\mathbf{a}_t \mid \mathbf{s}_t) P(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) \tag{1.2}$$

Goal-oriented RL

Tasks defined in this paper can be cast as goal-oriented RL problems, where the agent aims to reach a set of goal states $G = \{z\}$ predefined by the task, where $G \subseteq \mathscr{S}$ and z is absorbing. In this setting, a predicate [30] is defined to describe whether a goal state z is reached:

$$f_z : \mathscr{S} \to \{0,1\} \tag{1.3}$$

Therefore, the reward function $r(\mathbf{s}_t, \mathbf{a}_t)$ can be divided into two parts:

$$r(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} R & f_z(s_t) = 1 \\ r^c(\mathbf{s}_t, \mathbf{a}_t) & \text{otherwise} \end{cases} \tag{1.4}$$

where R is the success reward and function r^c the per-step rewards the agent may obtain during the course, e.g. reaching sub-goals, obstacle collision and time elapsed penalty.

Based on this general formulation, the reward functions for robotics tasks can be categorized into two types: those that consider r^c and those that do not. For instance, goal navigation tasks usually require r^c to count time elapsed when pursuing a shortest path to the goal. But in manipulation tasks such as opening a door, shaping the reward to guide policy optimization is usually hard and may limit the flexibility of the agent's behavior [30], thus, a binary reward indicating the success of a task is usually practical.

In the following sections, we will first introduce how to design our physics-inspired reward functions. Then we will demonstrate how to use them to enable our learned agents to be physics-aware in the proposed rearrangement tasks.

1.4 Physical Cost

1.4.1 Raw step physical cost

To teach an agent physical notions such as friction and mass in our pushing tasks, we compute the amount of physical cost $E(t)$ the agent spends on pushing an object to move on a floor at each time step t. $E(t)$ can be decomposed as the virtual physical work spent on translating and rotating the object:

$$E(t) = E_{trans} + E_{rot} \tag{1.5}$$

Assume that the object has mass m and the friction coefficient between the box and the floor is μ. The contact surface between the object and the floor has width X and length Y. The displacement

of the object's center of mass between consecutive time steps t and $t+1$ is Δx and the object rotates around its z-axis by an angle of θ.

The translation pushing cost can then be computed as:

$$E_{trans} = \mu mg\Delta x \tag{1.6}$$

where g is the gravitational acceleration. With $\phi = \arctan\frac{Y}{X}$, the rotation pushing cost is:

$$\begin{aligned} E_{rot} &= \mu\frac{mg}{XY}\theta\int_{-\frac{Y}{2}}^{\frac{Y}{2}}\int_{-\frac{X}{2}}^{\frac{X}{2}}\sqrt{x^2+y^2}dxdy \\ &= \mu\frac{mg}{XY}\theta\left[\frac{X^3}{12}\left(\frac{\sin\phi}{\cos^2\phi}+\ln\frac{1+\sin(\phi)}{\cos(\phi)}\right)\right. \\ &\left.-\frac{Y^3}{12}\left(\frac{-\cos\phi}{\sin^2\phi}+\ln\left|\tan\frac{\phi}{2}\right|\right)\right] \end{aligned} \tag{1.7}$$

where

$$\int_{-\frac{Y}{2}}^{\frac{Y}{2}}\int_{-\frac{X}{2}}^{\frac{X}{2}}\sqrt{x^2+y^2}dxdy = \frac{X^3}{12}\left(\frac{\sin\phi}{\cos^2\phi}+\ln\frac{1+\sin(\phi)}{\cos(\phi)}\right)-\frac{Y^3}{12}\left(\frac{-\cos\phi}{\sin^2\phi}+\ln\left|\tan\frac{\phi}{2}\right|\right) \tag{1.8}$$

and

$$\phi = \arctan\frac{Y}{X} \tag{1.9}$$

Note that all physical quantities involved here, the mass, friction coefficient, and the displacement and rotation of objects can either be obtained from the simulator or measured in the real-world (e.g. by GPS and compass). Thus, the proposed physical cost is applicable in both simulated and real environments regardless of what robot platform is used.

1.4.2 Normalized step physical cost

With raw step physical cost $E(t)$, we define the normalized step physical cost as

$$c(t) = \frac{E(t) - E_{min}(t)}{E_{max}(t) - E_{min}(t)} \tag{1.10}$$

where $E_{min}(t)$ and $E_{max}(t)$ are the running bounds:

$$\begin{aligned} E_{min}(t) &= \min\{E(0), \cdots, E(t)\} \\ E_{max}(t) &= \max\{E(0), \cdots, E(t)\} \end{aligned} \tag{1.11}$$

This normalized $c(t) \in [0,1]$ represents the pushing cost consumed at time step t relative to all history steps the agent has taken so far. For the edge case where $E_{min}(t) = E_{max}(t)$ which makes the denominator 0, we define $c(t) = 1$.

1.4.3 Normalized episode physical cost

Similarly, we define the normalized episode physical cost along an H-horizontal successful trajectory $\tau = (\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{s}_H, \mathbf{a}_H)$ as:

$$c(\tau) = \frac{E(\tau) - E_{min}(\tau)}{E_{max}(\tau) - E_{min}(\tau)} \tag{1.12}$$

where

$$E(\tau) = \sum_{t=0}^{H} E(t) \tag{1.13}$$

This episode physical cost is normalized over the episode physical cost of all successful episodes so far:

$$\begin{aligned} E_{min}(\tau) &= \min\{E(\tau_0), \cdots, E(\tau_T)\} \\ E_{max}(\tau) &= \max\{E(\tau_0), \cdots, E(\tau_T)\} \end{aligned} \tag{1.14}$$

Let $c(\tau) = 1$ when $E_{min}(\tau) = E_{max}(\tau)$.

1.5 Physics-aware Indoor Rearrangement Tasks

1.5.1 Environments

Our simulated 3D environments integrate Bullet physics engine [31], iGibson's renderer [2], and indoor scenes from the publicly available OpenRooms dataset [32]. OpenRooms provides 3D indoor scenes with ground truth material and real scene correspondence, allowing us to conduct robotics tasks with varying physical properties such as mass and friction coefficients. This setup also holds the potential for improving sim-to-real transfer. In this paper, the primary interactive environment used for learning is a classroom scene (6.95 m $\times$ 5.53 m $\times$ 3 m) that is sufficiently large to showcase differences in agent behavior. However, our tasks and methods are also applicable to other indoor scenes. The environment contains both static and fixed objects such as windows, doors, and ceiling lights, as well as K interactive boxes that the robot can move.

1.5.2 Agents

Our tasks are carried out by a simulated two-wheeled Fetch robot [33] using joint space control. Since the tasks only require locomotion, we keep joints such as the torso, arms, and grippers stationary and do not include them in the control process.

State space At each time step, the agent receives observations that include the current 3D positions and orientations of both the robot and interactive objects within the world Cartesian coordinate system. We use Euler angles to represent orientations. Therefore, for a task involving pushing K objects, the state is represented as a $6(K+1)$-dimensional vector. Notably, observing the velocities and accelerations of the robot and objects is not essential for task completion. Empirical results show that including these factors in the state space does not improve learning performance.

Action space The action space of our tasks is discrete and parameterized by v, which is the maximum value of wheel angular velocity (8.7 rad/s). Specifically, the robot has four actions: $[v, v]$ (move forward), $[0.5v, -0.5v]$ (turn left), $[-0.5v, 0.5v]$ (turn right) and $[0, 0]$ (stop).

1.5.3 Variable Friction Pushing Task

In this task, the agent and a box (10 kg) are initialized on a two-band floor. The red and blue bands have friction coefficients 0.2 and 0.8 respectively (Fig. 1.1). The agent's goal is to push the box to the target rectangular region located on the other band. Moving the box to any position inside the target region can be considered as a goal state.

Basic setting

To encourage learning efficiency, we design the reward function as:

$$r(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} R & \text{succeed} \\ r_n & \text{agent collides with obstacles} \\ r_p & \text{agent pushes box} \\ r_e & \text{otherwise} \end{cases} \tag{1.15}$$

where $R > 0$, $r_n < r_e < r_p < 0$, and r_e represents the time elapsed penalty. In practice, we set $R = 10$ and $r_n = -10, r_e = -1, r_p = -0.5$. On one hand, ranking the sub-rewards structures the task and provides a learning curriculum. On the other hand, this reward function makes the MDP a stochastic shortest path (SSP) setting [34], where the agent's objective is to reach the goal state while minimizing cumulative costs. In our case, this is equivalent to finding a policy for pushing the box to the target region as quickly as possible.

Physics-aware setting

To facilitate the agent's learning of different friction coefficients, we integrate the normalized step physical cost defined in (1.10) into the basic reward function (1.15), which results

in a physics-inspired reward function:

$$r(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} R & \text{succeed} \\ r_n & \text{agent collides with obstacles} \\ r_p \cdot c(t) & \text{agent pushes box} \\ r_e & \text{otherwise} \end{cases} \tag{1.16}$$

By weighting the pushing reward r_p with physical cost $c(t)$, the agent's optimization objective now becomes finding a policy to push the box to the target region in the most physically efficient way.

1.5.4 Variable Mass Pushing Task

In this task, the agent and two boxes are initialized in a circular region (with radius 1.5 m). The two boxes are instances of the same CAD model but with different materials and weights (10 kg and 50 kg, respectively). For a fair comparison, the robot's initial position is always at the center of the circle, while the two boxes are initialized at locations equidistant (0.99 m) to the robot.

To diversify the initial geometric configuration, we allow for four facing directions as the agent's initial orientation. For each facing direction, there are two choices to place the box: the light one on the left or on the right. Each training trial will correspond to a configuration covering all four directions. Therefore, we have 2^4 different configurations in total (Fig. 1.2). Then at the beginning of every episode, one facing direction is randomly selected from a certain configuration.

Basic setting

In the basic setting, the task succeeds when any one of the boxes is pushed outside the circular region. Unlike the variable friction pushing tasks, the time elapsed is not penalized in

this case:

$$r(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} R & \text{succeed} \\ r_n & \text{agent collides with obstacles} \\ 0 & \text{otherwise} \end{cases} \tag{1.17}$$

where $R > 0$ and $r_n < 0$. In practice, we set $R = 100$ and $r_n = -10$. Therefore, this is not an SSP problem. In other words, successful policies are equally optimal regardless of which box they choose to push or along which trajectory. The agent's decision on which box to push is expected to have no correlation with the box weight.

Physics-aware setting

To facilitate the agent's learning of the notion of mass, we integrate the normalized episode physical cost defined in (1.12) into the basic reward function (1.17) which results in a physics-inspired reward function:

$$r(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} R \cdot (1 - c(\tau)) & \text{succeed} \\ r_n & \text{agent collides with obstacles} \\ 0 & \text{otherwise} \end{cases} \tag{1.18}$$

Weighting the succeed reward R with the episode physical cost $c(\tau) \in [0, 1]$ informs the agent to choose a policy succeeding in task at the minimum physical cost. The optimal agent under this reward function will always choose to push the light box.

Note that reward structures (1.18) and (1.16) differ in the discount factor γ when encouraging physical efficiency. The episode physical cost $c(\tau)$ will not get discounted in (1.1) and thus is suitable for reward structure (1.18) which does not have a constant step reward. In contrast, the step physical cost $c(t)$ fits the reward structure (1.16). Incorrectly using $c(\tau)$ in (1.16) can lead to conflicting optimization objectives, as it would minimize the discounted cumulative time cost and the undiscounted episode physical cost at the same time.

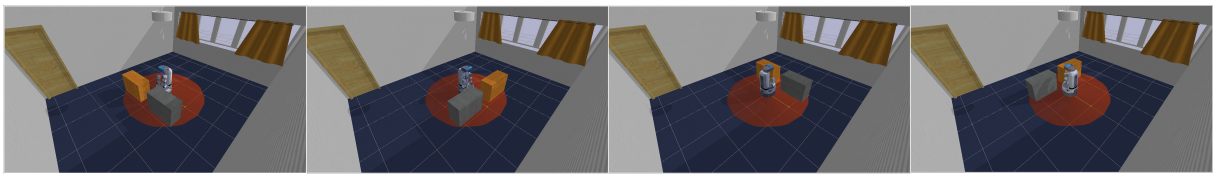


Figure 1.2. Illustration of configuration 0 in the variable mass pushing tasks. Swapping the light (orange) and heavy (gray) boxes in one to four directions gives the other 15 configurations.

1.5.5 Evaluation metrics

We use the mean episode physical cost $c(\tau)$ of successful trajectories (1.12) as the physical efficiency metric for the successful policies.

In the variable friction pushing task, we compute the fraction of the trajectory length spent on the low friction band for each evaluation episode. We denote this ratio as $\alpha(\tau) \in [0,1]$ and define it as:

$$\alpha(\tau) = \frac{l(\tau_{low})}{l(\tau)} \tag{1.19}$$

where τ is the total length of the box trajectory and τ_{low} is the length of the trajectory where the box is moved on the low-friction floor. The length of trajectory $l(\tau)$ is defined as

$$l(\tau) = \sum_{t=0}^{H-1} d(p_t, p_{t+1}) \tag{1.20}$$

where $d(p_t, p_{t+1})$ indicates the box's ℓ_2 displacement at time step t.

Based on this definition, the distribution of $\alpha(\tau)$ indicates the agent's preference of a slippery path to accomplish the pushing task. The distance between the two distributions with or without physics-inspired reward is then used as a measure of the agent's understanding of friction coefficient.

In the variable mass pushing task, we compute the histogram of how many heavy boxes an agent chooses to push over N configurations. The distance between the frequency distributions with or without physics-inspired reward is used as a measure of the agent's understanding of mass.

1.6 Experiments and Results

We evaluate our methods on the proposed tasks in both the basic setting and the physics-aware setting. The agents are trained using proximal policy optimization (PPO) [35] algorithms. We compare the policies learned by our physics-aware agent against the baseline agent and other cost computation methods to demonstrate the effectiveness of our physics-inspired reward function and the agent's awareness of physics notions. All of the agents share the same neural network architectures. The PPO hyperparameters KL coefficient $\{0.2, 0.3\}$ and clip parameter $\{0.3, 0.34, 0.35\}$ are tuned for different tasks. All other hyperparameters are the same.

The experiments are designed to answer the following questions:

1. Does our physics-inspired reward function correctly guide the agent to learn a physically efficient policy?

2. Does the policy learned under the physics-inspired reward function reflect the agent's awareness of mass and friction?

3. How does our definition of physical cost compare to other design strategies for learning successful and efficient policies?

1.6.1 Results of the variable friction pushing task

We first evaluate whether the agent can effectively learn a physically efficient policy for the variable friction pushing task when equipped with our physics-inspired reward function. We compare the learning curves of PPO with and without our physics-inspired reward (1.16) in Fig. 1.3. It is observed that the physics-aware agent (blue) converges to a policy with a clear drop in physical cost relative to the baseline agent (red). Although they both have a similar convergence rate, the learning curve of the physics-aware agent has a higher variance than the baseline agent before convergence. One possible explanation is that our adaptive physical cost is normalized by running bounds, which can lead to high variance in the per-step reward during the early stages of training, when the robot has limited knowledge of the environment.

In Fig. 1.4, we further visualize the trajectory distributions of physics-aware agent and the baseline agent during evaluation. Each agent is evaluated for 30 episodes. As discussed in Section 1.5.3, the agent is encouraged to find the quickest path to reach the goal region under the reward function without physical cost, which is approximately the shortest path between the initial box position and the bottom left corner of the goal region. Our results show that the empirical trajectory distribution is centered around this shortest path and spreads evenly toward the low-friction and high-friction areas. In contrast, the trajectories for the physics-aware agent consistently detour towards the low friction band due to its pursuit of physical efficiency.

We then compute the trajectory length ratio $\alpha(\tau)$ (Eq. 1.19) for each evaluation trajectory, and show the frequency distributions of evaluation trajectories in Fig. 1.6. The results show that our physics-aware agent concentrates on trajectories with very high $\alpha(\tau)$, which indicates its clear preference for the low friction trajectory. On the other hand, the baseline agent concentrates on $\alpha(\tau)$ around 0.5 which corresponds to trajectories with even traversal over low-friction and high-friction bands. The policy difference between the baseline and our physics-aware agents validates that the agents learned with the proposed physics-inspired reward are aware of friction coefficients.

1.6.2 Results of the variable mass pushing task

We now evaluate whether the proposed physics-inspired reward (1.18) can enable the agent to learn an physically efficient policy for the variable mass pushing task. We train agents over 10 configurations randomly picked from the total 16 configurations (Section 1.5.4), which is designed to prevent the agent from memorizing the correlations between box positions and masses. Each run of training is done in a certain configuration. At the beginning of each training episode, the agent is initialized to one of four facing directions in that configuration.

The learning curves over 10 configurations are shown in Fig. 1.5. In each run of training, the agent has to solve the *which to push* task at four facing directions. Accordingly, the learned policy can choose to push 0 to 4 heavy boxes, where 0 means that the agent never chooses the

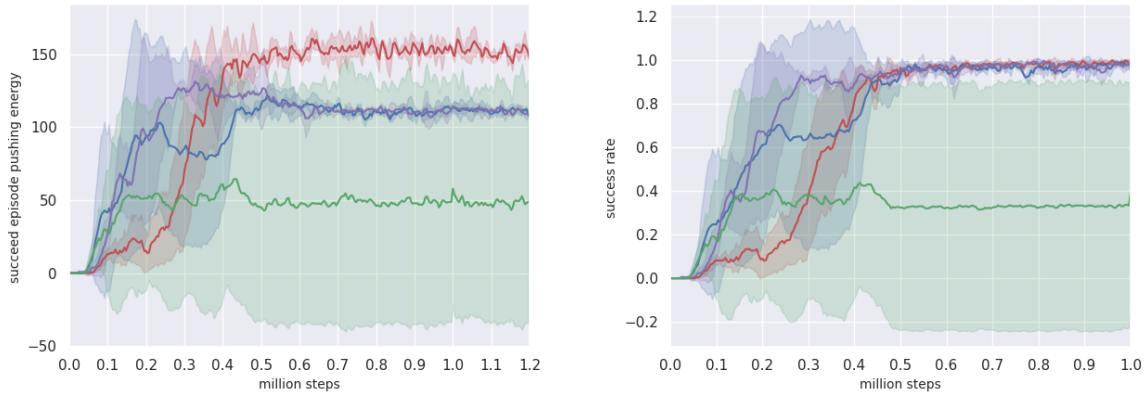


Figure 1.3. Training curves of PPO on variable friction pushing tasks. The algorithm is run across three different random seeds 1, 4, 8, and the shaded regions represent one standard deviation. Left: successful episode physical cost. Right: success rate. Red (No-physical-cost): baseline policy trained without physics-inspired rewards. Blue (Push-physics-running-bound / Ours): physics-aware policy trained with pushing cost rewards normalized by running bounds. Magenta (Push-physics-fix-bound): physics-aware policy trained with pushing physical cost normalized by fixed bounds. Green (Robot-physics-running-bound): physics-aware policy trained with robot physical cost normalized by running bounds.

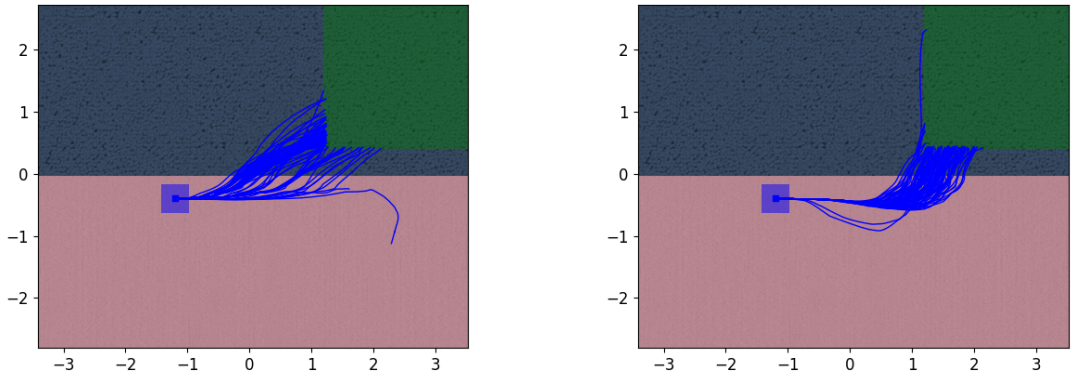


Figure 1.4. The distribution of 30 evaluation trajectories. Left: baseline agent trained without physics-inspired rewards. Right: physics-aware agent trained with our physics-inspired rewards. Pink band: low friction region. Blue band: high friction region. Blue square: the initial pose of the box at [-1.2m, -0.4m]. Green square: goal region (x=[1.2m, 3.53m], y=[0.4m, 2.72m]).

heavy box in all 4 directions. The policy that pushes 0 heavy boxes has a successful episode physical cost of around 40 in each run.

The results show that our physics-aware agent (blue) converges to a successful policy with episode physical cost concentrated around 40. This means that our physics-aware agent always chooses to push the light box for all facing directions and configurations. In contrast, the

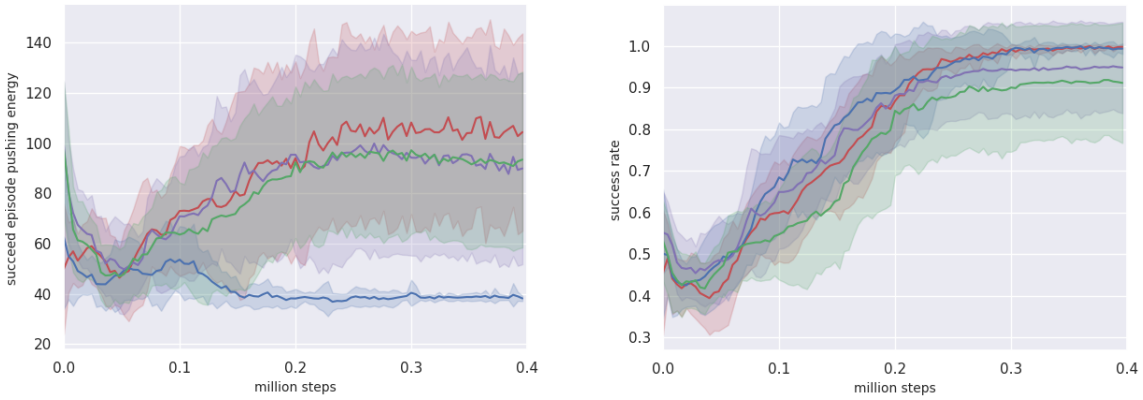


Figure 1.5. Training curves of PPO on variable mass pushing tasks, run across 10 configurations randomly selected out of 16, and the shaded regions represent one standard deviation. Left: successful episode physical cost. Right: success rate. Red (No-physical-cost): baseline policy trained without physics-inspired rewards. Blue (Push-physics-running-bound / Ours): physics-aware policy trained with pushing physics-inspired rewards normalized by running bounds. Magenta (Push-cost-fix-bound): physics-aware policy trained with pushing physical cost normalized by fixed bounds. Green (Robot-physics-running-bound): physics-aware policy trained with robot physical cost normalized by running bounds.

baseline agent (blue) with no physics-inspired reward has a successful episode physical cost that is uniformly distributed with a large standard deviation. This indicates that the baseline agent chooses to push the heavy box or light box almost randomly across different facing directions and configurations.

We then examine the frequency distribution of how many heavy boxes are pushed during the evaluation for each learned policy. Each agent is evaluated for 30 episodes under each of 10 configurations. The results are summarized in Fig. 1.7. We observe that our physics-aware agent highly favors pushing the light box in all 10 configurations, whereas the baseline agent chooses to push the light or heavy box with nearly equal probabilities. This contrasting behavior validates that, unlike the baseline agents, the agents learned with our proposed physics-inspired reward are aware of object masses.

1.6.3 Ablation study on physics-inspired reward design

We further compare our physics-inspired reward function with other design strategies:

1. Using robot output energy instead of task-relevant pushing cost defined in eq. (1.5).

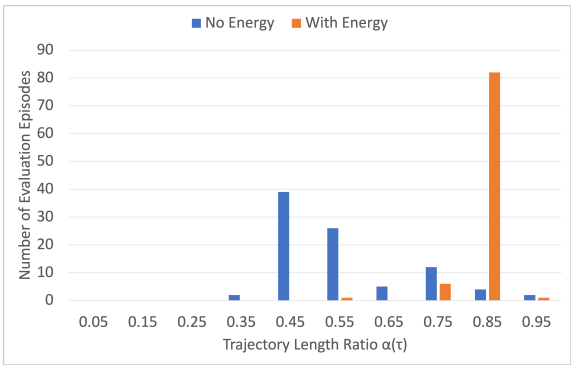


Figure 1.6. Frequency distributions of trajectory length ratio $\alpha(\tau)$ of 30 evaluation trajectories. Blue: baseline agent without physical cost. Orange: physics-aware agent trained with our physics-inspired reward.

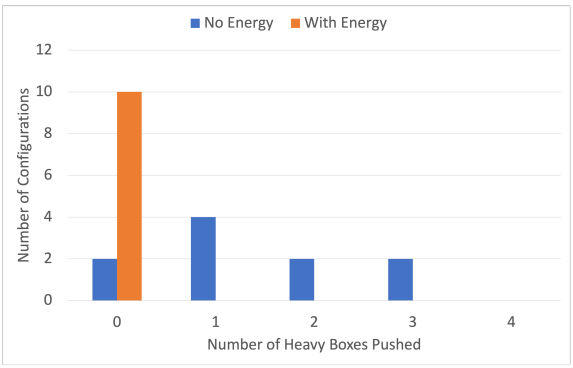


Figure 1.7. Frequency distributions of the number of heavy boxes pushed over 10 configurations. Blue: baseline agent without physical cost. Orange: physics-aware agent trained with our physics-inspired reward.

2. Normalized the physical cost by fixed bounds instead of running bounds defined in eq. (1.11) and (1.14).

The robot physical cost at time step t is the sum of the energy output of the robot's main driven joints (e.g. the left and right wheels of a Fetch robot). For joint j, its energy output during time step t is computed as

$$E(t,j) = |\omega_t^j \times \tau_t^j| \Delta t \tag{1.21}$$

where ω_t^j and τ_t^j are the angular velocity and torque of joint j, $\Delta t = 0.004s$ is the length of time interval. The fixed bounds of raw step pushing cost (1.5) is $[0, E_{max}]$ where

$$E_{max} = \mu_{max} m_{max} v_{max} g \Delta t \tag{1.22}$$

which involves the maximum object mass and friction coefficient in the scene and the robot's maximum moving velocity. The fixed bounds of normalized episode pushing cost (1.13) is $[0, E_{max} H_{max}]$ where $H_{max} = 400$ is the maximum time steps per episode.

The comparison of training curves is shown in Fig. 1.3 and 1.5. Using the robot physical cost normalized by running bounds leads to a drop in the success rate in both the variable friction and variable mass pushing tasks compared to our reward function. This is because the majority

of the robot output energy is spent on robot locomotion rather than pushing the boxes. Thus the normalized physical cost will encourage the ease of locomotion instead of accomplishing the pushing task.

When using the pushing cost normalized by fixed bounds, the success rate drops and the agent fails to find the physically efficient policy in the variable mass pushing task. This occurs because the number of time steps required for the agent to complete a task cannot be predicted before training. Using pre-defined fixed bounds restricts the physical cost to a narrow range and fails to account for the wide variation in costs across different successful trajectories. However, in the variable friction pushing task, where the cost is computed per step and the cost range is relatively small, there is little performance difference between using fixed or running bounds. Nonetheless, running bounds are preferable as they require less prior knowledge about the environment.

1.7 Conclusion and future work

We have studied the problem of how to make a learned agent physics-aware through a sequential decision-making process. We designed two novel indoor rearrangement tasks to train and evaluate the agent's policies in response to different masses and friction coefficients. To facilitate the learning, we introduced a novel physics-inspired reward function. Our results show that, compared to policies learned without this reward function, the agent can consistently learn to act with awareness of mass and friction differences.

Our work made a step towards learning physics-aware policies which are responsive to different masses and frictions. Potential future works include: (1) Extending the current method to vision-based RL setting for better generalization. (2) Transferring the physics-inspired policies, value functions, and experiences to downstream safe-critic tasks to help avoid risky behaviors such as falling down on slippery floor or pushing immovable objects.

Chapter 1, in part, is a reprint of the material as it appears in "Learning to Rearrange

with Physics-Inspired Risk Awareness", Meng Song, Yuhan Liu, Zhengqin Li, Manmohan Chandraker, Risk Aware Decision Making Workshop at The Robotics: Science and Systems, 2022. The dissertation author was the primary investigator and author of this paper.

Chapter 2

A Minimalist Prompt for Zero Shot Policy Learning

2.1 Introduction

Learning skills that are generalizable to unseen tasks is one of the fundamental problems in policy learning and is also a necessary capability for robots to be widely applicable in real-world scenarios. Recent works [36, 37, 38, 39] have shown that transformer based policies can be easily steered to solve new tasks by conditioning on task specifications (prompts). Tasks can be specified in various and complementary forms such as language instructions [36, 40], goal images [41, 42] and demonstrations [43, 44, 45]. Among typical task specifications, prompting with demonstration is one of the most expressive interfaces to communicate task objectives and human intent to the agent. Even a segment of demonstration contains rich information about environment dynamics, success states, and optimal behaviors to solve the task when expert-level demonstrations are provided. Demonstrations can also be represented in the original action and state space, which does not require any modality alignment. In addition, demonstrations are more favorable in describing goal locations and behaviors of low-level control than language instructions. However, specifying tasks via demonstrations suffers from two major limitations.

Firstly, it is impractical or unreasonable to have access to any form of demonstrations in test environments, or even refer to expert demonstrations of solving the test task beforehand. In some sense, directly providing the agent with demonstrations during deployment is akin to

revealing the solutions to the student in a test, thereby making the generalization meaningless. An appropriate task specification should be zero-shot, that is, conveying what to do instead of how to do it. For example, in a kitchen scenario, one may ask a robot to move a bottle onto the table. A natural task specification is to simply provide the robot with a description of the task rather than hinting at the possible trajectory of its arm and the target pose of its gripper when the task succeeds. This motivates us to distill the task description which is key to the generalization from a demonstration prompt and remove the skill-related information. Secondly, although a demonstration prompt encompasses rich information, there is little understanding of which factors are essential for the generalization. For example, given a visual demonstration of a running half-cheetah, it is unclear which factor contributes to the generalization – the appearance of the cheetah, the dynamics, the gait, or the desired speed.

To answer these questions, we consider the contextual RL setting where the training and test MDPs are assumed to be drawn from the same distribution which is controlled by a set of task parameters. These parameters can describe any properties of an MDP, which could either be physics-level attributes such as texture, location, velocity, mass, and friction, or semantic-level attributes, such as object color, shape, quantity, and category. These quantified task variations provide the necessary information for the agent to generalize across different tasks. Therefore, prompting an agent with these task parameters serves as an interpretable baseline for generalization. We show that conditioning a decision transformer on these task parameters alone generalizes on par with or better than its demonstration-conditioned counterpart. This suggests that a DT model implicitly extracts task parameters from the demonstration prompt to allow generalization to a new task.

Besides investigating the necessary role of task parameters in how the generalization happens, we are also interested in how much generic information can be elicited from the supervision to help generalization. To this end, we additionally introduce a learnable prompt to further enhance the generalization performance. The learnable prompt and the task parameter together compose a minimalist prompt for zero-shot generalization. We have empirically shown

that this proposed prompt has strong zero-shot generalization capability across a variety of control, manipulation, and navigation tasks.

In summary, the main contribution of this paper is a study of what is a minimalist prompt for decision transformers to learn generalizable policies on robotics tasks. (1) We show that a task parameter prompting decision transformer can generalize on par with or better than its state-of-the-art few-shot counterpart – the demonstration prompting DT in a zero-shot manner. This observation identifies the task parameter as necessary information for generalization rather than the demonstrations. (2) We propose to use an additional learnable prompt to extract the generic information from the supervision and show that this information can further boost the generalization performance.

2.2 Related Work

Generalization in policy learning Learning a policy that can be generalized to novel tasks has been extensively explored under different learning objectives and conditions. For example, meta-reinforcement learning [46] trains an RL agent to rapidly adapt to new tasks where the agent is enabled to continue to learn from its online interactions with the test environments. [47] trains a source RL policy operating in a disentangled latent space and directly transfers it to the visually different target domain. Recent works [48] extend meta-RL to an offline setting where the agent is fine-tuned on pre-collected transitions in test tasks for adaptation. Unlike these prior works focusing on optimizing the RL objective, we are interested in understanding the generalization behavior of a policy learned under an imitation learning objective, i.e. behavior cloning.

Few-shot generalization Generalization in the imitation learning setting can be categorized into two categories: few-shot and zero-shot generalization. Few-shot methods condition the model on demonstrations or example solutions from the test environments to enable generalization. For example, [43, 44, 40] conditions their model on the trajectory segments from expert

demonstrations of the test tasks. [42] solves the test task by matching the visual observations along the demonstration trajectories. [49, 45] requires a single demonstration to generalize to new tasks although the demonstration could be performed by an agent with different morphology. [41] transfers the skills learned from human videos to a new environment by conditioning on the goal image of the successful state. Learning a few-shot policy is akin to in-context learning [50], but differs in that it is not an inference-only method.

Zero-shot generalization In contrast, zero-shot methods assume no access to reference solutions in the test environments which is more realistic. Possible ways to specify a task in a zero-shot way include natural language instructions [51, 36, 52], driving commands [53], programs [54], etc. In this work, we demonstrate that task parameters, such as the target position of an object and the goal velocity, can sufficiently specify a task to enable zero-shot generalization.

Goal-conditioned policy Learning a goal-conditioned policy can also be viewed as a way of generalization. The *goal* here usually refers to a goal state indicating the success of a task. A goal-conditioned policy is hoped to reach any goal state including those unseen during training and can be solved as online RL [55, 30, 56], offline RL [57], and imitation learning problems [58, 59]. Mathematically, we can draw a connection between goal-conditioned policy and prompting a language model. We can think of the goal state as a special prompt when solving a goal-conditioned imitation learning problem with the policy instantiated as a transformer-based model.

Decision transformer Decision transformer [60] (DT) is one of the successful sequence modeling methods [61] in solving RL via supervised learning [62] problems. It combines the advantages of return-conditioned policy learning [63, 64, 62] and the expressive capability of GPT2. DT has been extended to solve offline multi-task learning [65, 66], online RL [67] problems. Complementary to the prior works of using DT in few-shot policy learning [43, 44], our method enables DT-based policy to generalize zero-shot to new tasks.

Adapting language models in the robotics domain Recent work has shown that pre-

trained large language models (LLMs) can be used in the robotics domain to ground skills and enable emergent reasoning via supplying high-level semantic knowledge to the downstream tasks [36, 68, 38]. In a similar spirit, our work can be viewed as training a language model agent from scratch to perform robotics tasks with essential task parameters and a learnable prompt as instructions.

Prompt learning A series of work [69, 70] in NLP has shown that a pre-trained language model can be adapted to downstream tasks efficiently via learning prefixes or soft prompts, which is called *prompt tuning*. In this paper, we adapt this technique to learn a generalizable policy which we refer to as *prompt learning*. We condition the decision transformer on a learnable prompt to extract the generic information for zero-shot generalization. Note that in our case, the prompt is learned jointly with the language model during training and both are frozen during inference, whereas in prompt tuning, the soft prompt is only trained during inference while the language model is frozen.

2.3 Problem Formulation

Contextual RL framework We study the problem of policy generalization under the framework of contextual RL [71], where each task T corresponds to a MDP denoted by a tuple $\mathscr{M} = (\mathscr{S}, \mathscr{A}, \mathscr{P}, \rho_0, r, \gamma)$. $\mathscr{S}$ denotes the state space, $\mathscr{A}$ denotes the action space, $\mathscr{P}(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ is a conditional probability distribution defining the dynamics of the environment. $\rho_0(\mathbf{s}_0)$ is the initial state distribution, $r : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$ defines a reward function, and $\gamma \in (0, 1]$ is a scalar discount factor.

In contextual RL setting, a policy is trained on N training tasks $\{T_i\}_{i=1}^{N}$ and tested on M unseen tasks $\{T_j\}_{j=1}^{M}$. All training and test tasks are assumed to be drawn from the same distribution over a collection of MDPs. Each MDP $\mathscr{M}(\mathbf{c})$ in the collection is controlled by a task parameter $\mathbf{c}$, which can parameterize either the dynamics $\mathscr{P}(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$, the reward function $r(\mathbf{s}_t, \mathbf{a}_t)$, the state space $\mathscr{S}$, and action space $\mathscr{A}$.

Contextual RL framework is expressive enough to formulate many generalization problems in policy learning. A typical example of such problems is to learn a goal-reaching policy [30, 59] that can reach any goal state $\mathbf{g} \in \mathscr{S}$. This can be formulated as training a goal-conditioned policy under an indicator reward function of having the current state exactly match the goal state. In this case, the task parameter $\mathbf{c}$ is the goal state $\mathbf{g}$. The definition of transfer learning in the contextual RL setting can also cover the cases where the generalization happens across different dynamics [72], state spaces [41], and action spaces [73, 74].

Behavior cloning (BC) in contextual RL setting In this paper, we consider the problem of learning a generalizable policy under the behavior cloning objective in a contextual RL setting. In this case, each training task T_i is paired with a dataset of demonstration trajectories $\mathscr{D}_i^{\text{train}} = \{\tau_k\}_{k=1}^K (i = 1, \cdots, N)$. The entire training set $\mathscr{D}^{\text{train}} = \{\mathscr{D}_i^{\text{train}}\}_{i=1}^N$ were pre-collected in each training environment and could be of any quality.

The objective of BC is to estimate a policy distribution $\hat{\pi}$ by performing supervised learning over $\mathscr{D}^{\text{train}} = \{(\mathbf{s}_i^\star, \mathbf{a}_i^\star)\}_{i=1}^n$:

$$\widehat{\pi} = \text{argmin}_{\pi \in \Pi} \mathscr{L}(\pi) = \text{argmin}_{\pi \in \Pi} \sum_{i=1}^{n} \mathscr{L}(\mathbf{s}_i^\star, \mathbf{a}_i^\star; \pi) \tag{2.1}$$

where we set the loss function $\mathscr{L}$ as squared ℓ_2 distance $\mathscr{L}(\mathbf{s}_i^\star, \mathbf{a}_i^\star; \pi) = \|\pi(\mathbf{s}_i^\star) - \mathbf{a}_i^\star\|_2^2$.

Evaluation metrics For a fair comparison, we follow the conventions in [75, 40] of evaluating the policy performance across different tasks. The performance is reported as a normalized score:

$$\text{normalized score } = 100 \times \frac{\text{return-random return}}{\text{expert return-random return}} \tag{2.2}$$

where 100 corresponds to the average return of per-task expert and 0 to the average return of a random policy.

The performance of a trained agent can be evaluated in two settings: (1) **Seen tasks**: The

agent is evaluated in each of N training tasks, which is consistent with the standard RL and IL evaluation protocols. (2) **Unseen tasks**: The agent is evaluated in each of M test tasks which are *strictly unseen* during training to measure its zero-shot generalization ability.

2.4 Approach

2.4.1 Decision transformer

Recently, there has been an increasing interest in casting policy learning problems as supervised sequential modeling problems [62, 61]. By leveraging the power of transformer architecture and hindsight return conditioning, Decision Transformer-based models [60, 65] have achieved great performance in offline policy learning problems.

Formally, Decision Transformer (DT) is a sequence-to-sequence model built on a transformer decoder mirroring GPT2 [76] architecture, which applies a causal attention mask to enforce autoregressive generation. DT treats an episode trajectory τ as a sequence of 3 types of input tokens: returns-to-go, state, and action, i.e. $\tau = \left(\widehat{R}_0, \mathbf{s}_0, \mathbf{a}_0, \widehat{R}_1, \mathbf{s}_1, \mathbf{a}_1, \ldots, \widehat{R}_{T-1}, \mathbf{s}_{T-1}, \mathbf{a}_{T-1}\right)$, where the returns-to-go $\widehat{R}_t = \sum_{i=t}^{T} r_i$ computes the cumulative future rewards from the current step until the end of the episode. At each time step t, it takes in a trajectory segment of K time steps $\tau_t = \left(\widehat{R}_{t-K+1}, \mathbf{s}_{t-K+1}, \mathbf{a}_{t-K+1}, \ldots, \widehat{R}_{t-1}, \mathbf{s}_{t-1}, \mathbf{a}_{t-1}, \widehat{R}_t, \mathbf{s}_t, \mathbf{a}_t\right)$ and predicts a K-step action sequence $A_t = (\hat{\mathbf{a}}_{t-K+1}, \ldots, \hat{\mathbf{a}}_{t-1}, \hat{\mathbf{a}}_t)$.

A DT parameterized by θ instantiates a policy $\pi_\theta(\mathbf{a}_t | \mathbf{s}_{-K,t}, \mathbf{a}_{-(K-1),t-1}, \widehat{R}_{-K,t})$, where $\mathbf{s}_{-K,t}$ denotes the sequence of K past states $\mathbf{s}_{\max(0,t-K+1):t}$. Similarly, $\widehat{R}_{-K,t}$ denotes the sequence of K past returns-to-go, and $\mathbf{a}_{-(K-1),t-1}$ denotes the sequence of $K-1$ past actions. By conditioning on a sequence of decreasing returns-to-go $\widehat{\mathbf{R}}_{-K,t}$, a DT policy becomes aware of its distance to the target state under a measure defined by the reward function.

Given an offline dataset of n K-length trajectory segments $\mathscr{D} = \{\tau_i^*\}_{i=1}^n$. The policy π_θ

is trained to optimize the following behavior cloning objective:

$$\min_\theta \mathscr{L}(\theta) = \min_\theta \sum_{i=1}^{n} \mathscr{L}(\tau_i^\star; \pi_\theta) \tag{2.3}$$

Specifically, we use the following squared ℓ_2 distance as the loss function in the continuous domain:

$$\mathscr{L}(\tau_i^\star; \pi_\theta) = \sum_{j=1}^{K} \| \pi_\theta(\mathbf{a}_j | \mathbf{s}^{i*}_{-K,j}, \mathbf{a}^{i*}_{-(K-1),j-1}, \widehat{R}^{i*}_{-K,j}) - \mathbf{a}^{i*}_j \|_2^2 \tag{2.4}$$

During the evaluation, a target returns-to-go $\widehat{R}_0$ should be specified at first. This represents the highest performance we expect the agent to achieve in the current task. At the beginning of an evaluation episode, DT is fed with $\widehat{R}_0$, $\mathbf{s}_0$ and generates $\mathbf{a}_0$, which is then executed in the environment and $\mathbf{s}_1$, r_1 are observed by the agent. We then compute $\widehat{R}_1 = \widehat{R}_0 - r_1$ and feed it as well as $\mathbf{s}_1$, $\mathbf{a}_1$ to the DT. This process is repeated until the episode ends.

2.4.2 Minimalist Prompting Decision Transformer

To identify the essential factors of a demonstration prompt in generalization and further extract generalizable information from the supervision, we propose a minimalist prompting Decision Transformer (Fig. 2.1). Given task parameter $\mathbf{c}$, a minimalist prompt consists of two parts: (1) A task parameter vector $[\mathbf{c}, \mathbf{c}, \mathbf{c}]$ (2) A learnable prompt $\mathbf{z} = [\mathbf{z_1}, \mathbf{z_2}, \mathbf{z_3}]$, where $\mathbf{z_i}$ $(i = 1, 2, 3)$ is a $n \times h$ dimensional vector corresponding to an embedding of n tokens. The number of $\mathbf{z_i}$ and $\mathbf{c}$ is designed to be three, paired with the three types of tokens $\widehat{R}, \mathbf{s}, \mathbf{a}$ in the input sequences.

A minimalist prompting DT represents a policy $\pi_\theta(\mathbf{a}_t | \mathbf{w}_t, \mathbf{c}; \mathbf{z})$, where context $\mathbf{w}_t$ denotes the latest K timestep input context at time step t. To train the policy on a task, the learnable prompt $\mathbf{z}$ is optimized together with the model weights θ over the associated training set $\mathscr{D} = \{\tau_i^*\}_{i=1}^n$,

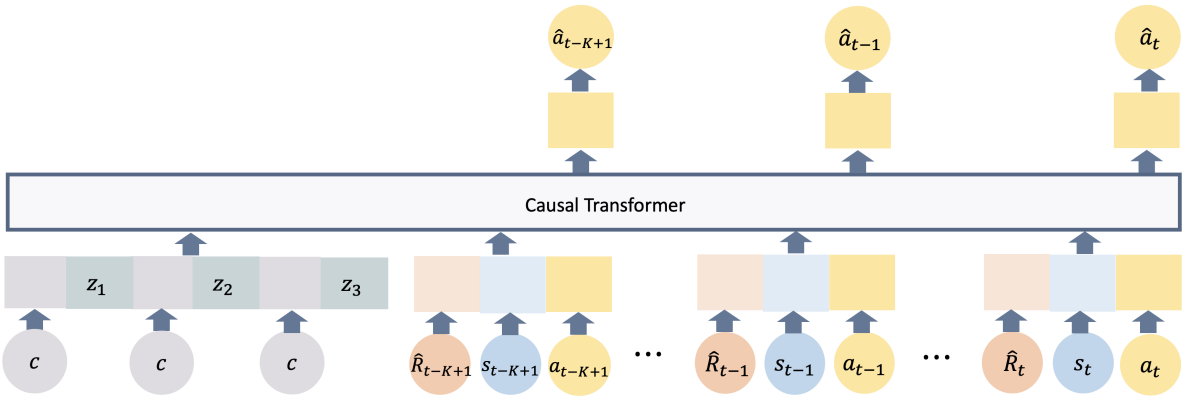


Figure 2.1. Architecture of the minimalist prompting decision transformer. At each time step t, the model receives the recent K time step input trajectory prepended by a minimalist prompt (gray and green part) and outputs a sequence of actions until $\hat{\mathbf{a}}_t$. A minimalist prompt consists of a task parameter vector $[\mathbf{c},\mathbf{c},\mathbf{c}]$ and a learnable prompt $\mathbf{z} = [\mathbf{z_1},\mathbf{z_2},\mathbf{z_3}]$.

thus the learning objective is

$$\min_{\theta,z} \mathscr{L}(\theta,\mathbf{z};\mathscr{D},\mathbf{c}) = \min_{\theta,z} \sum_{i=1}^{n} \mathscr{L}(\tau_i^\star;\pi_\theta(\cdot|\cdot,\mathbf{c};\mathbf{z})) \tag{2.5}$$

When training the policy $\pi_\theta(a_t|\mathbf{w}_t,\mathbf{c};\mathbf{z})$ across N tasks, the task parameter vector carries the task-specific information, whereas the learned prompt $\mathbf{z}$ captures the generic information shared by all training tasks. To see this, we can think of DT as a generator mapping latent variable $\mathbf{z}$ to actions, where $\mathbf{z}$ is adjusted to be broadly applicable to all of the training tasks. Therefore, although prompt $\mathbf{z}$ is **not allowed** to be optimized during the test time, it is expected to be transferable to any unseen tasks from the same task distribution.

2.4.3 Algorithms

We now show how a minimalist prompting DT is trained and evaluated in contextual RL setting in Alg. 1 and 2. Assume that we have N training tasks $\mathscr{T}^{train} = \{T_i^{\text{train}}\}_{i=1}^N$, each of them has an associated offline trajectory dataset $\mathscr{D}_i^{\text{train}} = \{\tau_j\}_{j=1}^m$ and a task parameter $\mathbf{c}_i^{\text{train}}$. For each gradient step, the minimalist-prompting DT is optimized according to (2.5) over a batch of trajectory segments and task parameters gathered from each training task.

We evaluate a trained minimalist-prompting DT for each of M test tasks $\mathscr{T}^{test} = \{T_i^{test}\}_{i=1}^{M}$ online. Since DT is inherently a return-conditioned policy, commanding the agent to perform a specific task should be conveyed as a desired target return G_i. To test the agent at its fullest capability, G_i is usually set as the highest task return in task i, which therefore requires prior knowledge about the test task. In practice, for each test task, G_i can be either set as the highest expert return or the highest possible return inferred from the upper bound of the reward function. With G_i as its desired performance, the policy performs online evaluations by interacting with the test environment for E episodes. The returns are then converted to the normalized score defined in (2.2) and averaged for across-task comparison.

Algorithm 1. Minimalist Prompting DT Training

Input: θ, $\mathbf{z}$ ▷ Initialized parameters

$\{\mathscr{D}_i^{\text{train}}, \mathbf{c}_i^{\text{train}}\}_{i=1}^{N}$ ▷ Offline data and task parameters from N training tasks

for each iteration **do**

for each gradient step **do**

$\mathscr{B} = \emptyset$, $\mathscr{U} = \emptyset$

for each task T_i **do** ▷ Sample a batch of data

$\mathscr{B}_i = \{\tau_j\}_{j=1}^{|B_i|} \sim \mathscr{D}_i^{\text{train}}$ ▷ Sample a mini-batch of K-length trajectory segments

$\mathscr{B} \leftarrow \mathscr{B} \cup \mathscr{B}_i$

$\mathscr{U} \leftarrow \mathscr{U} \cup \{\mathbf{c}_i^{\text{train}}\}_{|B_i|}$

end for

$[\theta, \mathbf{z}] \leftarrow [\theta, \mathbf{z}] - \alpha \nabla_{\theta,z} \mathscr{L}(\theta, \mathbf{z}; \mathscr{B}, \mathscr{U})$ ▷ Update model weights and learnable prompt

end for

end for

Output: θ, $\mathbf{z}$ ▷ Optimized parameters

2.5 Experiments and Results

In this section, we evaluate the zero-shot generalization ability of the minimalist prompting DT over a set of benchmark control and manipulation tasks and compare it with the demonstration-prompting counterpart. We start by introducing the experiment setting and then proceed to the result analysis.

Algorithm 2. Minimalist Prompting DT Evaluation

Input: $\theta, \mathbf{z}$ ▷ Trained policy
$\{\mathscr{M}_i = \left(\mathscr{S}_i, \mathscr{A}_i, \mathscr{P}_i, \rho_0^i, r_i, \gamma_i\right)\}_{i=1}^{M}$ ▷ MDPs of M test tasks
$\{\mathbf{c}_i^{\text{test}}\}_{i=1}^{M}$ ▷ Task parameters of M test tasks
$\{G_i^{\text{test}}\}_{i=1}^{M}$ ▷ Target returns-to-go of M test tasks
  for each task T_i **do**
    $\mathbf{c} \leftarrow \mathbf{c}_i^{\text{test}}$ ▷ Get task parameter
    $\widehat{R}_0 \leftarrow G_i^{\text{test}}$ ▷ Get task specific target returns-to-go
    for each test episode **do**
      $s_0 \sim \rho_i(s_0)$ ▷ Sample an initial state s_0
      Initialize context $\mathbf{w}_t$ with s_0, $\widehat{R}_0$
      for t=0, $T-1$ **do**
        $\mathbf{a}_t \leftarrow \pi_\theta(\mathbf{a}_t|\mathbf{w}_t, \mathbf{c}; \mathbf{z})$ ▷ Get an action $\mathbf{a}_t$ using the policy
        $\mathbf{s}_{t+1} \sim P_i(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ ▷ Execute the action $\mathbf{a}_t$ and observe $\mathbf{s}_{t+1}$, r_{t+1}
        $r_{t+1} = r_i(\mathbf{s}_t, \mathbf{a}_t)$
        if done **then**
          break
        end if ▷ Episode terminates
        $\widehat{R}_{t+1} = \widehat{R}_t - r_{t+1}$ ▷ Update returns-to-go
        $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t \| (\mathbf{a}_t, \widehat{R}_{t+1}, \mathbf{s}_{t+1})$ ▷ Update context
      end for
    end for
  end for

2.5.1 Environments and tasks

The experiments are conducted on two widely used benchmarks: control benchmark MACAW [48] for offline meta-reinforcement learning and manipulation benchmark Meta-world [77] for meta-reinforcement learning. Specifically, we evaluate our approach on the following five problems:

Cheetah-Velocity A simulated half-cheetah learns to run at varying goal velocities. The tasks differ in the reward functions $r_g(s,a)$ which is parameterized by the goal velocity g. We train in 35 training tasks and evaluate in 5 test tasks with unseen goal velocities. The half-cheetah has 6 joints and 20-dimensional state space which includes the position and velocity of the joints, the torso pose, and COM (Center of Mass) of the torso. Note that the goal velocity is not observable to the agent.

Ant-Direction A simulated ant learns to run in varying 2D directions. The tasks differ in the reward functions $r_g(s,a)$ which is parameterized by the goal direction g. We train in 45 training tasks and evaluate in 5 test tasks with unseen goal directions. The ant has 8 joints and 27-dimensional state space which is represented by the position and velocity of the joints and pose of the torso. Note that the goal direction is not observable to the agent.

By following the same data preparation methods in [48] and [43], the offline data for each task of Cheetah-Velocity and Ant-Direction are drawn from the lifelong replay buffer of a Soft Actor-Critic [78] agent trained on it, where the first, middle, and last 1000 trajectories correspond to the random, medium, and expert data respectively.

ML1-Reach, **ML1-Push**, **ML1-Pick-Place** A Sawyer robot learns to reach, push, and pick and place an object at various target positions. The tasks within each problem vary in the reward functions $r_g(s,a)$ and initial state distributions $\rho_o(s_0)$, where $r_g(s,a)$ is parameterized by the target position g and $\rho_o(s_0)$ is parameterized by the initial object position o. The state space is 39 dimensional including the pose of the end-effector and the pose of the object in the recent two time steps. Note that the target position is not part of the state. For each problem, there

are 45 training tasks and 5 test tasks with unseen target positions. The expert demonstration trajectories for each task are generated by running the provided scripted expert policy.

2.5.2 Baselines

To understand which factors contribute to generalization, we compare the full minimalist-prompting DT (Task-Learned-DT) with four baseline approaches, including three variants of minimalist-prompting DT, one state-of-art few-shot prompt DT method, and the original decision transformer:

Minimalist-prompting DT with both task parameter prompt and learned prompt (Task-Learned-DT) This is our proposed method in its fullest form. In the experiments, we use goal velocity, goal direction, and target position as the task parameter for Cheetah-Velocity, Ant-Direction, and ML1 tasks respectively.

Minimalist-prompting DT with task parameter prompt only (Task-DT) To study the effect of the learned prompt on zero-shot generalization, we remove it from Task-Learned-DT and only keep the task parameter prompt. This can be thought of as a special case of Task-Learned-DT where the learned token has length 0.

Pure-Learned-DT To study the effect of the task parameter prompt in zero-shot generalization, we remove it from Task-Learned-DT and only keep the learned prompt.

Trajectory-prompting DT (Trajectory-DT) [43] is a state-of-the-art few-shot policy learning method that shares a prompt DT architecture similar to ours. However, this method requires segments of expert demonstration trajectories as prompt in the test environments, which are unrealistic in most real-world applications.

Decision transformer (DT) We apply the original decision transformer [60] to our experiment setting. By excluding any prompts, we train the original decision transformer in multiple tasks and test it in other unseen tasks. This helps us understand the role of prompts in generalization.

2.5.3 Experimental results

We investigate the zero-shot generalization ability of Task-Learned-DT and identify the essential factors by comparing it to the baselines. The zero-shot generalization performance is evaluated online under the mean normalized score in unseen test tasks. If not mentioned explicitly, all methods are trained on the same expert dataset in each task. In particular, Trajectory-DT is conditioned on the expert trajectory prompts sampled from the corresponding expert dataset. For a fair comparison, all algorithms are run across the same three seeds. Our experiments aim to empirically answer the following questions:

How does minimalist-prompting DT generalize compare to its demonstration-prompting counterpart?

We compare Task-Learned-DT with the baselines on all five benchmark problems. The results are shown in Figure 2.2 and Table 2.1, in which Task-Learned-DT and Task-DT consistently outperform Trajectory-DT across all five problems and exceed it by a large margin on ML1-Pick-Place, ML1-Push, and Ant-Direction (Table 2.2). This indicates that the task parameters have precisely encoded sufficient task variations to specify the new task. Trajectory-DT tries to extract the same information from the prompting expert trajectories but performs worse than directly feeding it. In other words, prompting the problem solutions to the agent is inferior to prompting the task parameters.

To better understand how generalization happens via the proposed prompt, we further investigate the role of the learnable prompt. By comparing Task-Learned-DT and Task-DT, we find that the learned prompt improves generalization in most tasks, yet on ML1-Pick-Place and Cheetah-Velocity, adding a learned prompt slightly hurts the generalization (Table 2.2). However, we observed that the learned prompt will improve the generalization when tuned to an appropriate length. We thereby hypothesize that to benefit generalization, the length of the learned prompt should capture the dimension of the common representation of the tasks. In other words, one should choose a longer learned prompt if the solutions to different tasks share more

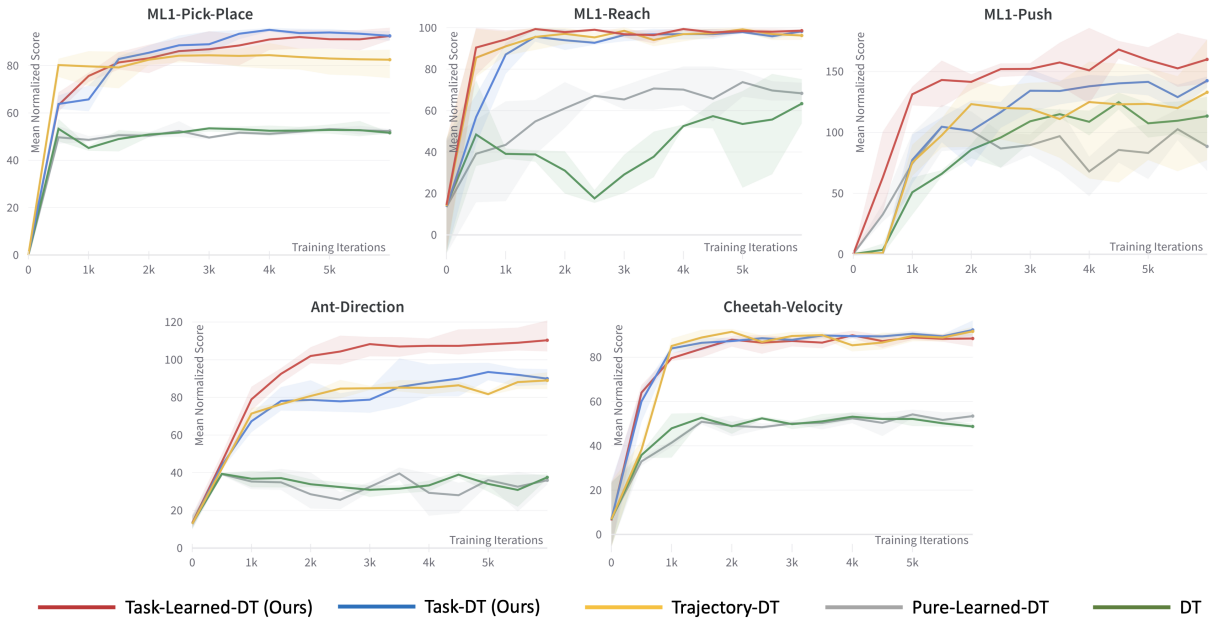


Figure 2.2. Comparing full minimalist-prompting DT (Task-Learned-DT) with four baselines: Task-DT, Trajectory-DT, Pure-Learned-DT and DT. The zero-shot performance of each algorithm is evaluated through the entire training process on five benchmark problems and reported under the mean normalized score. Shaded regions show one standard deviation of three seeds.

similarities. We will look into this later via ablating the length of the learned prompt in Table 2.3.

Next, we study the generalization ability provided by the task parameters by comparing Task-Learned-DT with Pure-Learned-DT and DT. The results show that Pure-Learned-DT performs similarly to DT and is significantly worse than Task-Learned-DT. In other words, learning a prompt without task parameters is almost equivalent to having no prompt at all. This implies that without the guidance of task parameter prompts, extracting the representation of a shared skill alone is not sufficient to perform well in a new task. In other words, the task-specific prompt is necessary for our method to outperform Trajectory-DT, although not always contribute to a larger performance gain.

How does the length of the learned prompt affect minimalist-prompting DT?

We vary the token number of the learned prompt in the range of $\{0, 3, 15, 30\}$ on ML1-Pick-Place, ML1-Push, and Ant-Direction to study its impact on generalization (Table 2.3). The

Table 2.1. Comparison on five benchmark problems

Problem	Task-Learned-DT (Ours)	Task-DT (Ours)	Trajectory-DT	Pure-Learned-DT	DT
ML1-Pick-Place	92.52 ± 3.03	$\mathbf{92.58 \pm 1.99}$	82.50 ± 6.73	52.39 ± 1.21	51.63 ± 0.22
ML1-Reach	$\mathbf{98.58 \pm 1.02}$	98.25 ± 1.04	96.24 ± 2.42	68.33 ± 4.49	63.35 ± 10.86
ML1-Push	$\mathbf{159.88 \pm 18.23}$	142.42 ± 4.89	132.91 ± 50.58	88.50 ± 25.90	113.39 ± 21.96
Ant-Direction	$\mathbf{110.35 \pm 9.10}$	89.96 ± 4.41	89.04 ± 4.43	36.00 ± 2.28	37.55 ± 1.36
Cheetah-Velocity	88.43 ± 3.15	$\mathbf{92.29 \pm 3.81}$	91.64 ± 1.83	53.42 ± 2.17	48.72 ± 1.02

Table 2.2. Performance improvement on five benchmark problems

Problem	max(Task-DT,Task-Learned-DT) - Trajectory-DT	Task-Learned-DT - Task-DT
ML1-Pick-Place	10.08	-0.06
ML1-Reach	2.34	0.33
ML1-Push	26.97	17.46
Ant-Direction	21.31	20.39
Cheetah-Velocity	0.65	-3.86

Table 2.3. Ablation on learned prompt length

Problem	0	3	15	30
ML1-Pick-Place	$\mathbf{92.58 \pm 1.99}$	91.31 ± 5.36	92.52 ± 3.03	82.66 ± 8.62
ML1-Push	142.42 ± 4.89	134.46 ± 14.46	140.16 ± 10.85	$\mathbf{159.88 \pm 18.23}$
Ant-Direction	89.96 ± 4.41	98.28 ± 8.30	$\mathbf{110.35 \pm 9.10}$	100.04 ± 5.26

learned prompt affects the performance to varying degrees depending on the problem domain. We observed that having a longer learned prompt does not always lead to better performance. A learned prompt of length 15 or 30 often achieves satisfactory performance even if not necessarily the best.

Can minimalist-prompting DT master multiple skills as well as performing zero-shot generalization?

To better understand the role of task parameters and the learned prompt, we consider a more challenging setting where we train the agent to master distinct skills and evaluate its zero-shot generalization to new tasks. Meta-world ML10 benchmark [77] provides 10 meta-task for training. Each of them corresponds to a distinct skill such as closing a drawer and pressing a button. Varying the object and goal positions within each meta-task yields 50 tasks. We then randomly draw 5 held-out tasks for testing in each meta-task and train on the others. We use the

Table 2.4. Comparison on ML10 tasks

Problem	Task-Learned-DT (Ours)	Task-DT (Ours)	Trajectory-DT
ML10	**74.56 ± 4.11**	57.85 ± 0.91	63.14 ± 5.65

same task parameter as ML1.

Table 2.4 compares zero-shot generalization performance of Task-Learned-DT, Task-DT, and Trajectory-DT in this setting. Among them, Task-Learned-DT outperforms Trajectory-DT by a large margin. This indicates that minimalist-prompting DT can act as a generalist agent with zero-shot generalization capability. Although the task parameters of each meta-task has a similar form, minimalist-prompting DT is able to interpret them according to the context environment and modulate itself to generalize to the corresponding new task. In addition, the performance gain of Task-Learned-DT over Task-DT indicates that the learned prompt becomes more necessary in this multi-skill generalization setting. We interpret this as mastering more skills requires grasping their commonalities better.

2.6 Limitations

As shown in Table 2.3, the length of the learned prompt is a problem-dependent hyper-parameter for our method to reach its best performance. Our approach also inherits limitations from Decision Transformer: (1) Since DT is optimizing a behavior cloning (BC) objective, it may suffer from a performance drop as the training trajectories become sub-optimal or even random [79] (Table B.2). (2) Deploying a DT-based model to a new task requires setting up a target returns-to-go in advance to reflect the desired highest performance, which needs some prior knowledge about the task.

2.7 Conclusion

In this work, we study the problem of what are the essential factors in the demonstration prompts for generalization. By identifying them, we hope to get rid of the assumption that

demonstrations must be accessible during deployment, which is unrealistic in most real-world robotics applications. Under the framework of contextual RL, we empirically identify the task parameters as the necessary information for a decision transformer to generalize to unseen tasks. Built on this observation, we additionally introduce a learnable prompt to further boost the agent's generalization ability by extracting the shared generalizable information from the training supervision. We demonstrated that the proposed model performs better than its demonstration prompting counterpart across a variety of control, manipulation, and planning benchmark tasks.

Chapter 2, in full, is a reprint of the material as it appears in "A Minimalist Prompt for Zero-Shot Policy Learning", Meng Song, Xuezhi Wang, Tanay Biradar, Yao Qin, Manmohan Chandraker, Task Specification Workshop at The Robotics: Science and Systems, 2024. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Probabilistic World Modeling with Asymmetric Distance Measure

3.1 Introduction

Learning good representations from the data plays a crucial role in the success of modern machine learning algorithms [80]. It requires an AI system to have the ability to extract rich structures from the data and build a model of the world. What structures should be preserved, summarized, and what should be abstracted out is heavily dependent on the downstream tasks and learning objectives. For example, visual representation learning for recognition tasks mainly aims to expose the clustering structures in the representation space. Generative models such as VAEs [81], GANs [82], BERT [83], diffusion models [84] and autoregressive models such as Transformers and RNNs [85, 86] capture the compressed information necessary for reconstructing the data from its corrupted version, or for directly generating the future predictions based on the data. Manifold learning methods such as LLE [87] and ISOMAP [88] aim to learn a low-dimensional representation space preserving the local or global geometric structure of the data manifold. Interchangeably using one for another would not yield the best performance. In this paper, we consider the problem of *what is a good representation for planning and reasoning in a stochastic world*. We will derive the problem formulation by delving into the definitions of planning and reasoning, along with the stochastic nature of the world.

Planning and reasoning typically involve an optimization process finding the most proba-

ble future outcomes, the most reasonable answers, or the most effective path to the goal. This optimization ability sets it apart from retrieving and interpolating good solutions seen in the training data, which imitation learning algorithms are good at. Instead, it enables problem-solving in new ways. *To allow for such optimization, a fundamental requirement for the representation space is to have a distance function that accurately measures the probability of an event occurring given another, or the distance from the current state to the goal state.* For example, the optimal value function $V^*(\mathbf{s}, \mathbf{g})$ in goal-conditioned RL, representing the optimal expected return when an agent is starting from state $\mathbf{s}$ and aiming to reach a particular goal state $\mathbf{g}$, can serve as a distance measure [89]. However, a substantial body of previous work [90, 91] has focused on learning the dynamics through a generative model that directly predicts the next state. Generating a predicted state on a detailed level is computationally expensive, especially when dealing with high-dimensional observations, and is more necessary for simulation than for planning. Furthermore, it is unclear whether the representation space learned by such reconstruction-based methods has a distance measure that adequately supports the optimization process in planning.

The world inherently operates as a stochastic dynamical system, transiting from one state to another, often formulated as a stochastic process such as a Markov chain. A Markov chain induces a directed transition graph whose edges represent one-step transition probabilities from one state to another. By leveraging this transition graph, we can estimate the probability of transitioning from state $\mathbf{x}$ to state $\mathbf{y}$ within C time steps, which we refer to as *C-step reachability from* $\mathbf{x}$ *to* $\mathbf{y}$. Instead of looking at the single-step transition between neighbor states, C-step reachability considers all possible paths within C steps from $\mathbf{x}$ to $\mathbf{y}$. A path on the transition graph can hold various interpretations in different contexts. For example, it can represent a sequence of events leading from event $\mathbf{x}$ to event $\mathbf{y}$. It can represent a reasoning chain starting from evidence or assumption $\mathbf{x}$ to conclusion $\mathbf{y}$. It can also represent a sequence of problem-solving steps beginning with problem $\mathbf{x}$ and resulting in the final solution $\mathbf{y}$. Therefore, C-step reachability allows us to do multi-way probabilistic inference and answer questions such as "Given that event $\mathbf{x}$ has already occurred, how probable is it for event $\mathbf{y}$ to occur in the future, considering all

possible ways the future could unfold?", "How likely to reach **y** from **x** considering all possible paths?"

By integrating the above principles, we formulate the representation learning problem for planning and reasoning in a stochastic world as the task of learning an embedding space. This space's distance function reflects the state reachability induced by a Markov chain. To address this challenge, we first establish a formal connection between the one-step transition matrix and the C-step reachability, then encode C-step reachability into an asymmetric similarity function by binary NCE [92, 93]. A significant volume of prior studies on representation learning in NLP [94], computer vision [95, 96, 97] and RL [98, 99] have implicitly modeled the co-occurrence statistics on an undirected graph and embed it by a single mapping function. In contrast to these prior works, we use a pair of embedding mappings to model the dual roles of a state in a directed graph: as an outgoing vertex and an incoming vertex. This approach assures an asymmetric similarity function reflecting the asymmetric even irreversible transition probabilities (e.g. the transition of food from raw to cooked), which is crucial for planning and event modeling tasks.

Furthermore, the learned representation space also provides a geometric abstraction of the underlying directed graph in a perspective-dependent way. We find that by conditioning on a common reference state, a symmetric distance measure can be recovered to measure the point density in the representation space. This naturally gives rise to the notion of subgoals, which denote the geometrically salient states that only a handful of paths can lead through. The directional nature of the underlying transition graph reveals that the subgoal is inherently a relative concept and subject to change, which aligns well with our everyday experience but has not been explored by the previous works [100, 101]. For example, at a theme park entrance, a ticket is required for entry, but not for exit. Thus, the entrance serves as a subgoal upon arrival but not upon leaving. After formally defining the reference state conditioned distance measure, we can identify subgoal states as low-density regions using any density-based clustering algorithms. We demonstrate the effectiveness of our approach on solving the subgoal discovery task in a variety of gridworld environments.

3.2 Preliminaries

3.2.1 Markov chain and the directed transition graph

A Markov chain on state space $\mathscr{S}$ can be thought of as a stochastic process traversing a directed graph where we start from vertex $\mathbf{s}_0 \sim \rho(S_0)$ and repeatedly follow an outgoing edge $\mathbf{s}_t \to \mathbf{s}_{t+1}$ with some probabilities. We denote the transition probability distribution of the Markov chain as $P(S_{t+1}|S_t)$ $(t = 0, 1, \dots)$. Thus, a Markov chain induces a weighted directed graph $G = (V, E, P)$ which is called the *transition graph*. $V = \{\mathbf{s} \in \mathscr{S}\}$ is the vertex set, and $E = \{\mathbf{s} \to \mathbf{s}' \mid \mathbf{s}, \mathbf{s}' \in \mathscr{S}\}$ is a set of directed edges where each edge $\mathbf{s} \to \mathbf{s}'$ has probability $P(S_{t+1} = \mathbf{s}'|S_t = \mathbf{s})$ as its weight. In this work, we consider the induced transition graph in the most general setting where loops are allowed, and a directed edge does not have to be paired with an inverse edge.

3.2.2 MDP and the environment graph

A Markov decision process (MDP) $\mathscr{M} = (\mathscr{S}, \mathscr{A}, P, r, \gamma, \rho)$ is an extension of a Markov chain with the addition of actions and rewards. The induced transition graph G still has the states $\mathbf{s} \in \mathscr{S}$ as its vertex set, but the transition probability $P(S_{t+1} = \mathbf{s}'|S_t = \mathbf{s})$ from vertex $\mathbf{s}$ to its neighbor $\mathbf{s}'$ now involves a two-stage transition process: we move from $\mathbf{s}$ to $\mathbf{s}'$ through some actions $\mathbf{a}$ according to both the environment dynamics $P(S_{t+1}|S_t, A_t)$ and the agent's policy $\pi(A_t|S_t)$:

$$P^{\pi}(S_{t+1} = \mathbf{s}'|S_t = \mathbf{s}) = \int P(S_{t+1} = \mathbf{s}'|S_t = \mathbf{s}, A_t = \mathbf{a})\pi(A_t = \mathbf{a}|S_t = \mathbf{s})d\mathbf{a} \tag{3.1}$$

where the policy probability $\pi(A_t = \mathbf{a}|S_t = \mathbf{s})$ acts as a weight of the environment transition probability $P(S_{t+1} = \mathbf{s}'|S_t = \mathbf{s}, A_t = \mathbf{a})$.

In particular, when the policy is a uniform distribution for any state $\mathbf{s}$, (3.1) is irrelevant

to the policy π, i.e.

$$P(S_{t+1} = \mathbf{s}'|S_t = \mathbf{s}) = \frac{1}{|\mathscr{A}|}\int P(S_{t+1} = \mathbf{s}'|S_t = \mathbf{s}, A_t = \mathbf{a})d\mathbf{a} \tag{3.2}$$

We term this policy-agnostic transition graph as the *environment graph* since its edge weights encode the environment dynamics unbiasedly and the graph can be fully induced from an MDP. In addition, we ignore the rewards as they are task-specific and do not necessarily reflect the structure of the environment.

3.3 Problem formulation

Suppose that a Markov chain $\mathscr{M}$ on state space $\mathscr{S}$ has an initial distribution $\rho(S_0)$ and an **unknown** transition probability distribution $P(S_{t+1}|S_t)$. G is the transition graph induced from $\mathscr{M}$. Given a set of T-step sequences drawn from M, i.e. $\mathscr{T} = \{(\mathbf{s}_0^i, \mathbf{s}_1^i, \ldots, \mathbf{s}_T^i)\}_{i=1}^N \sim \mathscr{M}$, our goal is to learn state representations whose distance function reflect the asymmetric vertex reachability on the underlying G.

In many cases, the state space $\mathscr{S}$ is either unknown or uncountable. As a result, enumerating the states at the very beginning of a Markov chain is infeasible. Therefore, we do not require a uniform initial distribution like the previous works [102, 98, 99]. On the contrary, we study the typical cases where the initial distribution $\rho(S_0)$ is highly concentrated, either a narrow Gaussian or a δ distribution centered at a specific state $\mathbf{s}_0$.

3.3.1 Vertex reachability

Let random variables U and W represent any vertex on a directed graph $G = (V, E, P)$. The reachability from vertex $\mathbf{u}$ to vertex $\mathbf{w}$ can be defined as

$$P(W = \mathbf{w}|U = \mathbf{u}) = \lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T} P_t(S_t = \mathbf{w}|S_0 = \mathbf{u}) \tag{3.3}$$

$$
\begin{aligned}
P_t(S_t = \mathbf{w} | S_0 = \mathbf{u}) &= \sum_{S_1,\ldots,S_{t-1}} P(S_1,\ldots,S_{t-1},S_t = \mathbf{w} | S_0 = \mathbf{u}) \\
&= \sum_{S_1,\ldots,S_{t-1}} \prod_{i=0}^{t-1} P(S_{i+1}|S_i)
\end{aligned}
\tag{3.4}
$$

where the edge $S_i \to S_{i+1}$ is in E for all $0 \leq i \leq t-1$. $P(S_{i+1}|S_i)$ is the one-step transition probability distribution of G. In other words, the sequence of vertices $S_0 \to S_1 \to \cdots \to S_t$ is a t-step walk on G starting from $\mathbf{u}$ to $\mathbf{w}$. $P_t(S_t = \mathbf{w}|S_0 = \mathbf{u})$ is the probability of traveling from $\mathbf{u}$ to $\mathbf{w}$ in exactly t steps. The reachability from $\mathbf{u}$ to $\mathbf{w}$ can be understood as the likelihood of reaching $\mathbf{w}$ from $\mathbf{u}$ within any number of steps. The derivation is done by thinking of G as a Bayesian network. By definition, the reachability from vertex $\mathbf{u}$ to $\mathbf{w}$ may differ from the reachability from vertex $\mathbf{w}$ to $\mathbf{u}$, i.e. $P(W = \mathbf{w}|U = \mathbf{u}) \neq P(W = \mathbf{u}|U = \mathbf{w})$.

3.3.2 C-step approximation

Computing the vertex reachability is often intractable since it requires enumerating all possible paths over a near-infinite horizon. In practice, we approximate (3.3) by looking ahead C steps in a Markov chain. Formally, suppose that we have a T-step Markov chain $M = \{S_0, S_1, \ldots, S_T\}$ with transition probability distribution $P(S_{t+1}|S_t) \triangleq P$, $t = 0, 1, \ldots, T-1$. We denote any preceding random variables in the chain as the random variable $Y = \{S_i \mid 0 \leq i \leq T-1\}$, and any subsequent random variables within C steps from Y as the random variable $X = \{S_j \mid i < j \leq \min(i+C, T)\}$. The joint distribution $P(X = \mathbf{w}, Y = \mathbf{u})$ represents the occurrence frequency of **the ordered pair** $(\mathbf{u}, \mathbf{w})$ on all the possible paths within C steps. The reachability from $\mathbf{u}$ to $\mathbf{w}$ can be approximated as

$$
\begin{aligned}
P(W = \mathbf{w}|U = \mathbf{u}) &\approx P(X = \mathbf{w}|Y = \mathbf{u}) \\
&\approx \frac{1}{C} \sum_{t=1}^{C} (P^t)_{\mathbf{uw}}
\end{aligned}
\tag{3.5}
$$

where P^t denotes the t-step transition probability distribution which is the product of t one-step transition matrices P, i.e. $P^t = \underbrace{PP\cdots P}_{t}$. $(P^t)_{\mathbf{uw}} = P(S_t = \mathbf{w} \mid S_0 = \mathbf{u})$. $P(X|Y)$ is also a stochastic matrix where each row sums to one and represents the reachability distribution starting from a specific state.

The first approximation results from the fact that we reduce the near-infinite look-ahead steps to C steps. The second line holds when $0 < C \ll T$ (See Appendix C.1 for the proof). In this case, $P(X|Y)$ represents the accumulated transition probabilities up to C steps, which align with the definition of reachability in (3.3). This suggests that when $1 \leq C \ll T$ is satisfied, one should use a larger C to achieve better approximation precision.

3.4 Asymmetric contrastive representation learning

We now aim to learn representations whose distance function encodes the asymmetric state reachability distribution $P(X|Y)$. In this section, we show how this problem can be formulated and addressed within the framework of noise contrastive estimation (NCE) [103].

NCE is a family of powerful methods for solving the density estimation problem while avoiding computing the partition function. The core idea of NCE is to transform the density estimation problem into a contrastive learning problem between the target distribution and a negative distribution. Based on different objective functions, NCE methods fall into two categories: binary NCE which discriminates between two classes and ranking NCE which ranks the true labels above the negative ones for the positive inputs [93]. Although the ranking NCE is broadly used in the recent resurgence of contrastive representation learning [104, 95, 105, 96], we adopt binary NCE in this paper since it establishes a direct relationship between the scoring function and probability ratio.

3.4.1 Conditional binary NCE

Suppose that there is a data set of N trajectories drawn from a T-step Markov chain $\mathscr{M}$, i.e. $\mathscr{T} = \{(\mathbf{s}_0^i, \mathbf{s}_1^i, \ldots, \mathbf{s}_T^i)\}_{i=1}^N \sim \mathscr{M}$. We construct the positive data set for binary NCE

as $D^+ = \{(\mathbf{x}_i^+, \mathbf{y}_i) \sim P(X,Y)\}_{i=1}^{N^+}$, and negative data set as $D^- = \{\mathbf{x}_i^- \sim P_n(X)\}_{i=1}^{N^-}$. $P(X,Y)$ denotes the aforementioned joint distribution of preceding random variable Y and subsequent random variable X. $P_n(X)$ denotes a specified negative distribution on state space $\mathscr{S}$. We use K to denote the ratio of negative and positive samples, i.e. $N^- = KN^+$.

This yields a binary classification setting where a classifier $P(C|X, Y = \mathbf{y}; \theta)$ is trained to discriminate between positive samples from the conditional distribution $P(X|Y = \mathbf{y})$ and the negative samples from $P_n(X)$, $\forall \mathbf{y}$. We use the label $C = \{1, 0\}$ to denote the positive and negative classes, respectively. The training objective of binary NCE is to correctly classify both positive samples D^+ and negative samples D^- using logistic regression:

$$\begin{aligned}
&\max_{\theta} J_{BI_NCE}(\theta) \\
=&\max_{\theta} \mathbb{E}_{\mathbf{x}^+, \mathbf{y} \sim P(X,Y)} \log P(C = 1|\mathbf{x}^+, \mathbf{y}; \theta) + K\mathbb{E}_{\mathbf{y} \sim P_Y(Y)} \mathbb{E}_{\mathbf{x}^- \sim P_n(X)} \log P(C = 0|\mathbf{x}^-, \mathbf{y}; \theta) \quad (3.6) \\
=&\max_{\theta} \mathbb{E}_{\mathbf{x}^+, \mathbf{y} \sim P(X,Y)} \log \sigma(f_\theta(\mathbf{x}^+, \mathbf{y})) + K\mathbb{E}_{\mathbf{y} \sim P_Y(Y)} \mathbb{E}_{\mathbf{x}^- \sim P_n(X)} \log(1 - \sigma(f_\theta(\mathbf{x}^-, \mathbf{y})))
\end{aligned}$$

where $P_Y(Y)$ denotes the marginal distribution of Y. $\sigma(\cdot)$ denotes the logistic function.

When the classifier is Bayes-optimal, we have

$$\exp(f(X, Y = \mathbf{y})) = \frac{P(X|Y = \mathbf{y})}{KP_n(X)}, \quad \forall \mathbf{y} \tag{3.7}$$

where $f(X, Y = \mathbf{y})$ is the scoring function. (Proof in Appendix C.2)

3.4.2 Asymmetric encoders

One of our key observations is that the scoring function $f(X = \mathbf{x}, Y = \mathbf{y})$ in the conditional case of binary NCE can be treated as a similarity function between the embeddings of $\mathbf{x}$ and $\mathbf{y}$. Given that the underlying graph of a Markov chain is *directed*, the reachability from vertex $\mathbf{x}$ to

vertex $\mathbf{y}$ often differs from the reachability from vertex $\mathbf{y}$ to vertex $\mathbf{x}$. That is,

$$P(X = \mathbf{x}|Y = \mathbf{y}) \neq P(X = \mathbf{y}|Y = \mathbf{x}) \tag{3.8}$$

Therefore, having an asymmetric similarity function becomes essential to mirror the inherent asymmetry in the reachability probabilities. In other words, the function form of f should allow

$$f(X = \mathbf{x}, Y = \mathbf{y}) \neq f(X = \mathbf{y}, Y = \mathbf{x}) \tag{3.9}$$

One effective approach to accomplish this is by utilizing two separate encoders. Concretely, we use an encoding function $\phi : \mathscr{S} \to \mathscr{Z}$ to map the preceding random variable Y to the embedding space Z and use another encoding function $\psi : \mathscr{S} \to \mathscr{Z}$ to map the subsequent random variable X to the same embedding space.

$$f(X = \mathbf{x}, Y = \mathbf{y}) = s(\psi(\mathbf{x}), \phi(\mathbf{y})) \tag{3.10}$$

$s(\cdot, \cdot)$ could be an arbitrary similarity measure in space Z, e.g. cosine similarity or negative l_2 distance.

In this paper, we opt for cosine similarity as measure $s(\cdot, \cdot)$ in space Z because it is well-bounded and facilitates stable convergence.

$$s(\psi(\mathbf{x}), \phi(\mathbf{y})) = \frac{\psi(\mathbf{x})}{\|\psi(\mathbf{x})\|_2} \cdot \frac{\phi(\mathbf{y})}{\|\phi(\mathbf{y})\|_2} \tag{3.11}$$

In this setup, (3.7) becomes

$$\exp(\frac{\psi(\mathbf{x})}{\|\psi(\mathbf{x})\|_2} \cdot \frac{\phi(\mathbf{y})}{\|\phi(\mathbf{y})\|_2}) = \frac{P(X = \mathbf{x}|Y = \mathbf{y})}{KP_n(X = \mathbf{x})} \tag{3.12}$$

Employing two separate encoders offers two advantages over a single encoder. Firstly,

it guarantees the asymmetry of $f(\cdot,\cdot)$ irrespective of the choice of similarity measure $s(\cdot,\cdot)$ in the embedding space. Secondly, it ensures the asymmetry in a general sense without imposing constraints on the relationship between $\phi(\cdot)$ and $\psi(\cdot)$. On the contrary, prior works [104, 105, 106] employ a single encoder $\psi(\cdot)$ alongside a linear transformation A to model time-series data, which can be viewed as a specific case of our approach where $\phi(\cdot) = A\psi(\cdot)$.

With slight adjustments, our method can also *treat the directed graph as an undirected graph*. In such cases, we modify the range of random variable Y and X to $Y = \{S_i \mid 0 \leq i \leq T\}$ and $X = \{S_j \mid \max(i-C,0) \leq j \leq \min(i+C,T), j \neq i\}$. Additionally, we use a single encoder $\phi(\cdot)$ to encode both X and Y. $s(\cdot,\cdot)$ could be any symmetric similarity measure, e.g. cosine similarity. As a result, we have

$$\begin{aligned} f(X=\mathbf{x},Y=\mathbf{y}) &= f(X=\mathbf{y},Y=\mathbf{x}) \\ &= s(\phi(\mathbf{x}),\phi(\mathbf{y})) \\ &= \log \frac{P(X=\mathbf{x}|Y=\mathbf{y}) + P(X=\mathbf{y}|Y=\mathbf{x})}{2KP_n(X=\mathbf{x})} \end{aligned} \tag{3.13}$$

We transform the directed graph into an undirected one by enforcing the similarity function $f(\cdot,\cdot)$ to encode the average of $P(X=\mathbf{x}|Y=\mathbf{y})$ and $P(X=\mathbf{y}|Y=\mathbf{x})$. Note that the reachability from $\mathbf{x}$ to $\mathbf{y}$ and the reachability from $\mathbf{y}$ to $\mathbf{x}$ may not be identical, meaning $P(X=\mathbf{x}|Y=\mathbf{y})$ may not be equal to $P(X=\mathbf{y}|Y=\mathbf{x})$.

In fact, word embedding methods such as word2vec [94] and graph embedding methods such as node2vec [102] have implicitly performed the operations in (3.13) through the Skip-gram model. Learning undirected representations is sufficient when the downstream tasks are clustering, classification, etc. These applications rely on mutual similarity or compatibility measures insensible to the directionality. However, incorporating transition direction into representation learning is crucial for planning, reasoning, and event modeling. This becomes especially important when the transition is irreversible due to temporal order or causalities. For example, consider two states: $\mathbf{y}$=young and $\mathbf{x}$=old, transiting from young to old is certain while

the reverse is impossible. In probabilistic language, we can express it as $P(X = \mathbf{x}|Y = \mathbf{y}) = 1$ and $P(X = \mathbf{y}|Y = \mathbf{x}) = 0$. The expected embeddings of these two states should correctly reflect this difference in reachability. Ideally, the state 'young' should be very close to the state 'old' in one scenario and be infinitely far away in the other. Averaging these two probabilities into 0.5 would erroneously pull them together to the same distance in both cases.

3.4.3 The choice of the negative distribution

Although the NCE framework holds for an arbitrary negative distribution $P_n(X)$, different choices of $P_n(X)$ affect the similarity function $f(X,Y)$ in a meaningful way. We experimented with several choices of the negative distribution $P_n(X)$: $P_X(\cdot)$ - the marginal distribution of X, $P_Y(\cdot)$ the marginal distribution of Y, and $U(X)$ - the uniform distribution of X. Among these options, $P_X(\cdot)$ is the only choice that consistently performs well across various values of chain length T and step size C. In contrast, $U(X)$ performs the worst in most cases.

It is worth noting that when $P_n(X) = P_X(X)$, the similarity function $f(X,Y)$ encodes the pointwise mutual information (PMI) of an ordered pair $(\mathbf{y},\mathbf{x})$ up to a constant offset.

$$\begin{aligned} f(X = \mathbf{x}, Y = \mathbf{y}) &= \log \frac{P(X = \mathbf{x}|Y = \mathbf{y})}{KP_X(X = \mathbf{x})} \\ &= \log \frac{P(X = \mathbf{x}|Y = \mathbf{y})}{P_X(X = \mathbf{x})} + \log \frac{1}{K} \end{aligned} \tag{3.14}$$

where $\frac{1}{P_X(X=\mathbf{x})}$ can be viewed as a weight of the reachability $P(X = \mathbf{x}|Y = \mathbf{y})$, representing the inverse of the overall frequency of visits to the state $\mathbf{x}$. In other words, distribution $P(X = \mathbf{x}|Y = \mathbf{y})$ is adjusted according to the rarity of the arrival state $\mathbf{x}$.

3.5 Inference and planning using the learned asymmetric similarity function

Unlike standard representation learning methods which map each input into a single embedding vector, our approach maps each state $\mathbf{s}$ into two embedding vectors $\phi(\mathbf{s})$ and $\psi(\mathbf{s})$

in space Z. These two representations of $\mathbf{s}$ correspond to the two roles of a vertex in a directed graph respectively: $\phi(\mathbf{s})$ corresponds to the role of $\mathbf{s}$ being an outgoing vertex and $\psi(\mathbf{s})$ corresponds to the role of $\mathbf{s}$ being an incoming vertex. From the perspective of edges, each pair of vertices $\mathbf{u}$ and $\mathbf{v}$ can have two directed paths between them, $\mathbf{u} \to \mathbf{v}$ and $\mathbf{v} \to \mathbf{u}$, which have similarities $s(\phi(\mathbf{u}), \psi(\mathbf{v}))$ and $s(\phi(\mathbf{v}), \psi(\mathbf{u}))$ in the embedding space respectively. Therefore, the trained encoder $\phi(\cdot)$ and $\psi(\cdot)$ alongside the function $s(\cdot,\cdot)$ can fully capture the structure of the underlying directed graph G in C-step approximation.

After training $\phi(\cdot)$, $\psi(\cdot)$, we can use them to perform inference tasks on a given set of states χ. For instance, suppose that we are currently at state $\mathbf{s}$ and want to know which state is most likely to occur in the future. We can then identify it by finding the state closest to $\mathbf{s}$ in the latent space, i.e. solving $\arg\max_{\mathbf{x}\in\chi} s(\phi(\mathbf{s}), \psi(\mathbf{x}))$. Furthermore, with a defined action space, we can construct a directed graph on χ by evaluating $s(\phi(\cdot), \psi(\cdot))$. Feeding G to any search algorithm, such as Dijkstra's algorithm, enables us to plan a shortest path from an initial state $\mathbf{s}_0$ to a goal state $\mathbf{s}_g$ in the latent space.

3.6 Reference state conditioned distance measure

Building upon the learned asymmetric similarity function $s(\cdot,\cdot)$, we can still recover symmetric distance measures by conditioning on a reference state. This allows us to perform clustering in the latent space without worrying about the metric disagreement. The ambiguity in metrics arises because each pair of vertices in a directed graph has two directional distances. To resolve this ambiguity, there must be a criterion to select one of them instead of simply averaging. Based on this principle, we demonstrate that a metric consensus in a directed graph can be achieved by comparing each pair of states to a common reference state $\mathbf{r}$. Therefore, we term it as a *reference state conditioned distance measure*. Formally, given a reference state $\mathbf{r}$, we define the pairwise latent distance between $\mathbf{u}$ and $\mathbf{v}$ with respect to $\mathbf{r}$ as $d_{\mathbf{r}}(\mathbf{u}, \mathbf{v})$:

- If $s(\phi(\mathbf{r}), \psi(\mathbf{u})) \geq s(\phi(\mathbf{r}), \psi(\mathbf{v}))$,

$$d_{\mathbf{r}}(\mathbf{u}, \mathbf{v}) = 1 - s(\phi(\mathbf{u}), \psi(\mathbf{v})) \tag{3.15}$$

- Otherwise,

$$d_{\mathbf{r}}(\mathbf{u}, \mathbf{v}) = 1 - s(\phi(\mathbf{v}), \psi(\mathbf{u})) \tag{3.16}$$

In other words, for a pair of states $\mathbf{u}$ and $\mathbf{v}$, we always choose the one closer to $\mathbf{r}$ as the starting state and choose the other one as the ending state to evaluate their distance. The symmetry of $d_{\mathbf{r}}(\cdot,\cdot)$ follows by the definition, i.e. $d_{\mathbf{r}}(\mathbf{u},\mathbf{v}) = d_{\mathbf{r}}(\mathbf{v},\mathbf{u})$, $\forall \mathbf{v}, \mathbf{u}$. The reference state $\mathbf{r}$ could be any state in the state space $\mathscr{S}$, resulting in $|\mathscr{S}|$ symmetric distance measures $d_{\mathbf{r}}(\cdot,\cdot)$ extracted from $s(\cdot,\cdot)$. Unlike its undirected graph counterpart, there is no single and universal distance measure in a directed graph; instead, the measure varies depending on the observer's perspective. This definition of a reference-dependent measure $d_{\mathbf{r}}(\cdot,\cdot)$ allows us to find the salient geometric structure of the representation space from changing perspectives, which is meaningful in many real-world applications. We will demonstrate its usefulness in decision-making scenarios in the next section.

3.7 Subgoal discovery

Subgoal is a fundamental concept introduced to tackle planning or decision-making problems using the divide-and-conquer strategy. It refers to the intermediate objectives or states that an agent aims to achieve on the way to reaching the ultimate goal, and are often used to break down complex or long-horizon tasks into shorter, simpler, more manageable parts.

While lacking a unified mathematical formulation, previous works have proposed various definitions of subgoals, each emphasizing its different roles in dividing the overall task, which include: midway states between the starting and goal state that are reachable by the current policy [107, 89]; common states shared by successful trajectories [108, 109]; functionally salient

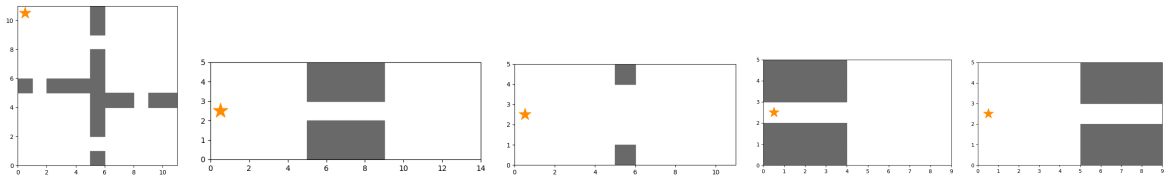
states where policy distributions significantly change [98]; and decision states where the goal state is informative to the decisions [100].

In this work, we propose to use the concept of subgoals to denote the key states that only a handful of actions can lead through. Intuitively, the number of paths passing through a subgoal would decrease significantly. Following the definition, these states should be intrinsic to the geometric characteristics of the transition graph, independent of specific tasks and goals. More importantly, since the transition graph is inherently directed, subgoals may change as the observer's perspective varies. Therefore, unlike the previous works, we consider the subgoals as relative and subject to change, rather than as fixed and absolute entities. More precisely, we identify subgoals as the states that cause a sharp decrease in pairwise reachability, as perceived from the perspective of the agent's current state.

By the proposed representation learning method, we can convert the reachability defined on the original state space into the point density in the embedding space. The reduction in reachability can subsequently be translated into the identification of low-density regions, which density-based clustering algorithms can readily solve. In practice, we perform DBSCAN [110] on the representations to group the closely packed points while identifying the outliers that lie alone in the low-density regions as subgoals.

3.8 Experiments

We evaluate our representation learning method in five gridworld environments with various layouts: Four rooms, Dumbbell, Wide door, Flask, and Nail as shown in Figure 3.1. These environments are designed to encompass basic geometric configurations, serving as building blocks for constructing larger environmental graphs. The states are 2D locations of the agent, $\mathbf{s} \in \mathbb{R}^2$. The action space includes five actions: left, right, up, down, and stop. In each environment, a data set of N trajectories $\mathscr{T} = \{(\mathbf{s}_0^i, \mathbf{s}_1^i, \ldots, \mathbf{s}_T^i)\}_{i=1}^N$ is collected by a uniform policy starting from a fixed initial state $\mathbf{s}_0$.



(a) Four rooms **(b)** Dumbbell **(c)** Wide door **(d)** Flask **(e)** Nail

Figure 3.1. Gridworld environments: The grey areas indicate the walls. The yellow star indicates the initial state $\mathbf{s}_0$.

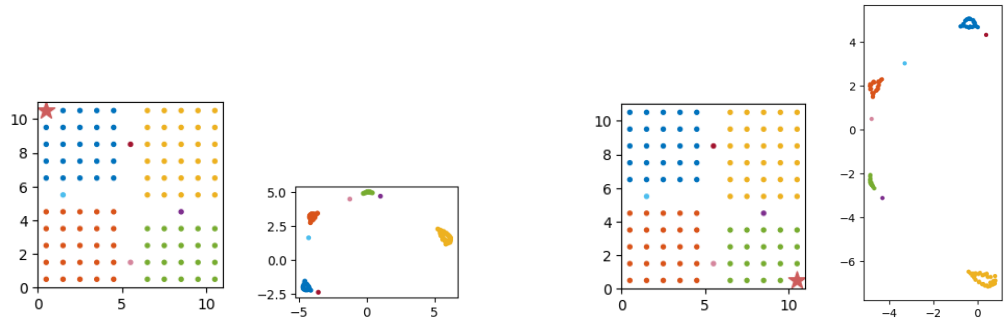
We train two encoders ϕ and ψ on $\mathscr{T}$ according to the learning objective (3.6). Both ϕ and ψ are parameterized as a 2-layer MLP with latent space $\mathbf{z} \in \mathbb{R}^{64}$. We set the ratio of negative and positive samples $K = 1$ throughout all the experiments. We found that $K > 1$ performs worse because the classification becomes imbalanced. Moreover, $K = 1$ also yields a precise correspondence between the learned asymmetric similarity function and PMI by removing the offset in (3.14). We use the marginal distribution of X as the negative distribution for training, i.e. $P_n(X) = P_X(X)$, which we found performs the best.

3.8.1 t-SNE visualization of the learned representations

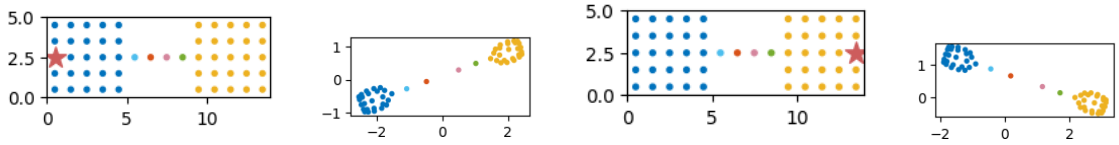
To examine the learned representations, we show the visualization of the learned representations and the original states in Figure 3.2. We use t-SNE to project the 64-dimensional learned representations onto 2D plots based on the reference state conditioned distance measure $d_{\mathbf{r}}(\cdot,\cdot)$ defined in (3.15) and (3.16). Across all the environments, the state space can be categorized into high-reachability regions such as rooms, and low-reachability regions such as long passages and various doorways (bottlenecks). Our learned representations can distinctly separate these regions according to their reachability with respect to the given reference state.

3.8.2 Subgoal discovery results

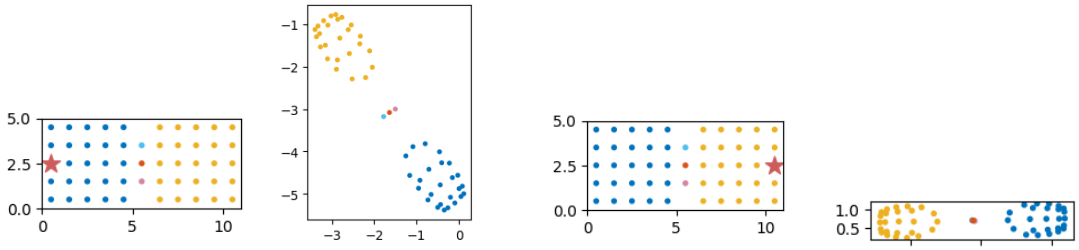
We now demonstrate that our learned distance measure $d_{\mathbf{r}}(\cdot,\cdot)$ enables easy identification of the subgoal states. We perform the DBSCAN clustering in the representation space according to $d_{\mathbf{r}}(\cdot,\cdot)$. The DBSCAN algorithm works by considering each point and expanding clusters



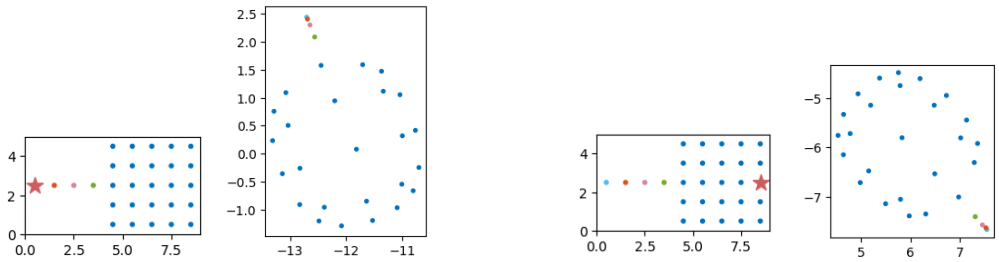
(a) Four rooms



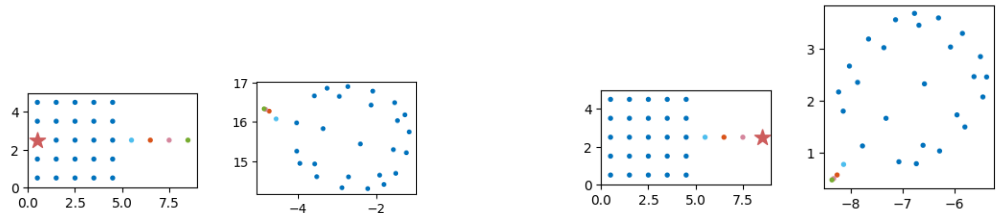
(b) Dumbbell



(c) Wide door



(d) Flask



(e) Nail

Figure 3.2. Visualization of the original states and the learned representations. The states are colored to visualize their position correspondences between two spaces. In each environment, the representation space is visualized from two different perspectives. The reference states **r** are indicated by the red stars. In each group, the left subplot shows the original states in 2D Euclidean space and the right subplot shows the t-SNE projection of the learned representations.

from densely populated areas. During this procedure, the distance measure $d_{\mathbf{r}}(\cdot,\cdot)$ is evaluated to compute a point's local density and connectedness. Once cluster expansion is completed, the remaining points are labeled as noise which are just the subgoals we aim to identify. Figure 3.3 shows the subgoals and clusters found by our algorithm in each environment. The low-reachability regions such as doorways and passages have been successfully identified as subgoals and the rooms are decomposed into clusters.

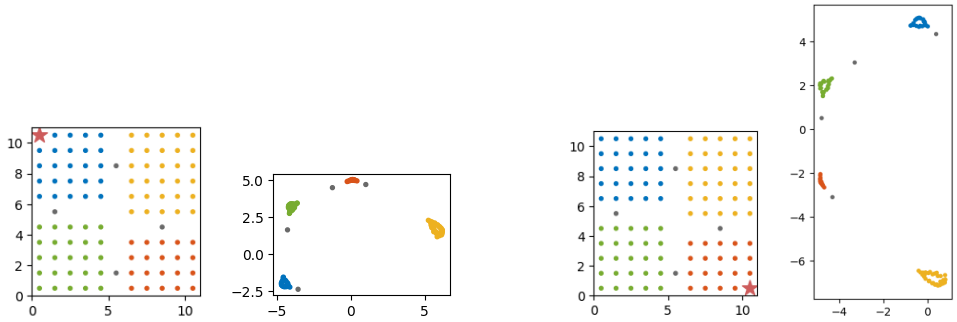
3.8.3 Ablation studies

The effect of approximation step size C

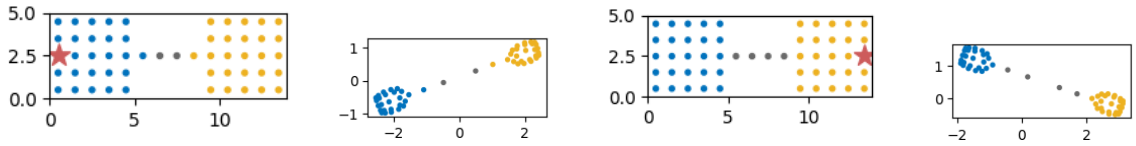
To better understand the effect of step size C in approximating the true reachability defined in (3.3), we compare the representations learned with $C = 1, 16, 64$ in the four rooms environment. When $C = 1$, the reachability is equivalent to the one-step transition probabilities. As the value of C increases, the C-step reachability considers more and more possible paths between two states and thus progressively approaches the true reachability. As a result, the abstraction level of the representations becomes higher. This trend has been successfully captured in the learned distance measure $d_{\mathbf{r}}(\cdot,\cdot)$. As shown in Figure 3.4, the rooms and doorways become further apart in the representation space as C goes up. This is because the reachability contrast becomes sharper as longer paths are considered. These empirical observations align well with our theoretical derivations on the influence of step size C in Section 3.3.2.

The choice of negative distribution

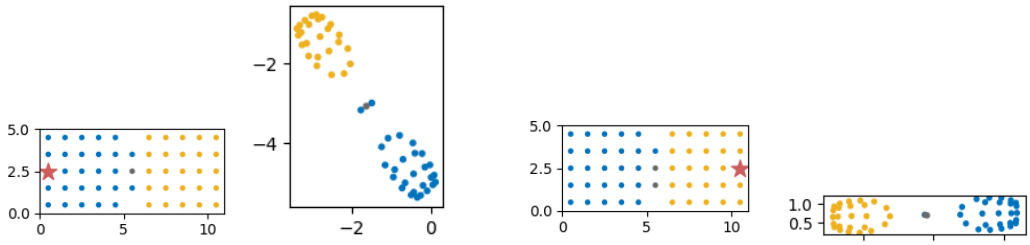
We evaluate the representations learned with negative distributions $P_X(X)$, $P_Y(X)$ and $U(X)$ in the Four Rooms environment in Figure 3.5 and 3.6. We observe that $P_X(X)$ consistently performs well across different values of episode length T and step size C while $U(X)$ performs the worst. We provide two hypotheses of why this phenomenon arises: (1) Using a negative distribution resembling the positive distribution helps train the classifier to reach Bayes optimum. $P_X(X)$ is more similar to $P(X|Y=\mathbf{y})$ than $U(X)$. (2) The neural networks are secretly weighting



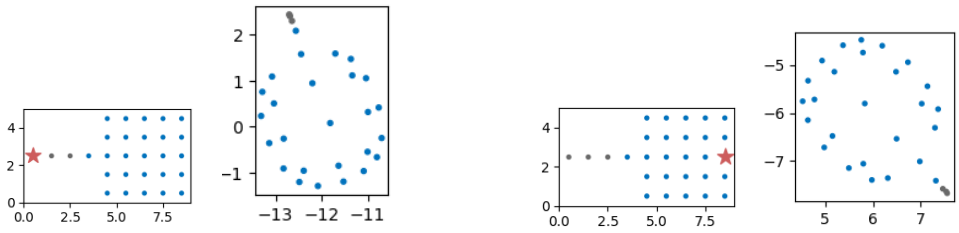
(a) Four rooms



(b) Dumbbell



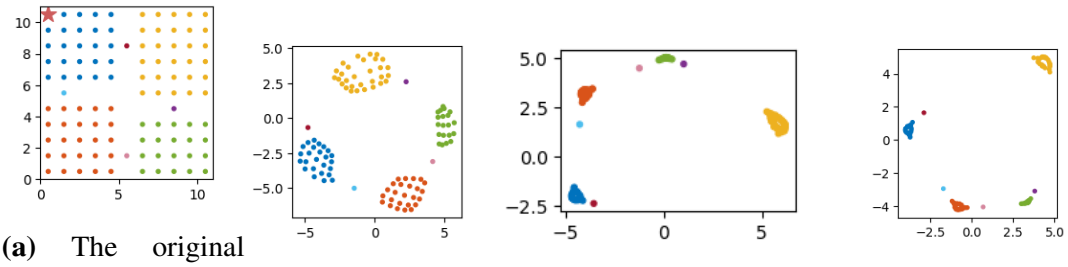
(c) Wide door



(d) Flask

(e) Nail

Figure 3.3. Subgoal discovery results. The states are colored according to the cluster labels in both the original space and the learned representation space. The gray states are subgoals. In each environment, we visualize the clustering results from two different perspectives. The reference states **r** are indicated by the red stars. In each group, the left plot shows the original states in the 2D Euclidean space and the right plot shows the t-SNE projection of the learned representations.



(a) The original states **(b)** $C = 1$ **(c)** $C = 16$ **(d)** $C = 64$

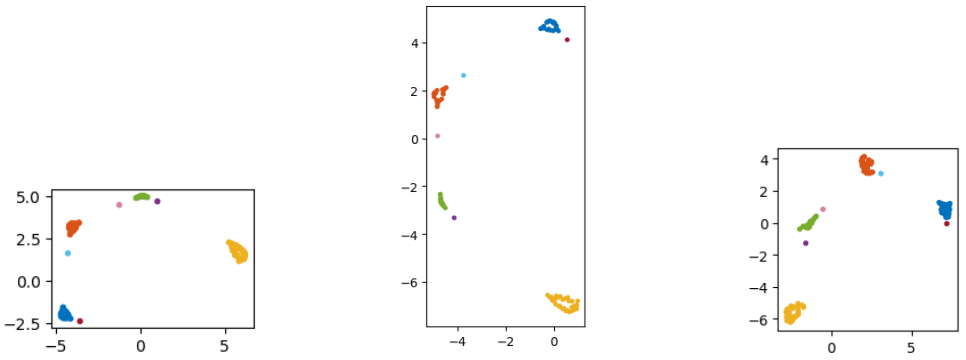
Figure 3.4. Visualization of the original states and the learned representations with different approximation step sizes C in Four Rooms environment. The embeddings are projected to 2D plots by t-SNE. In all the experiments, we train the encoders on a single episode of length $T = 153600$.

the probability of $\mathbf{x}$ by its overall visitation frequencies when counting the occurrences of $\mathbf{x}$ given $\mathbf{y}$. The weight acts as an "impression" of a state. Therefore, choosing $P_X(X)$ as the denominator rather than $U(X)$ can counteract this weighting effect and recover the true $P(X|Y = \mathbf{y})$.

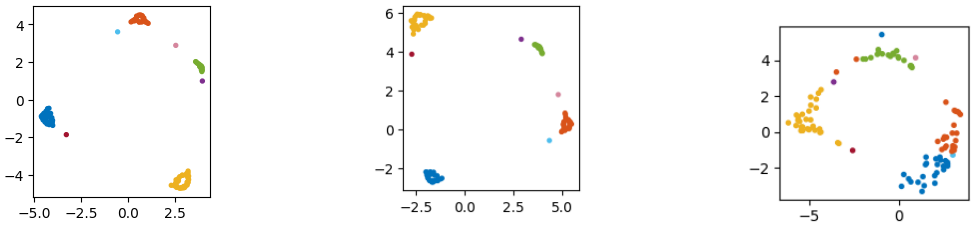
3.9 Conclusion and future work

In this work, we study the problem of what is a good representation for planning and reasoning in a stochastic world and how to learn it. We discussed the importance of learning a distance measure in allowing planning and reasoning in the representation space. We modeled the world as a Markov chain and introduced the notion of C-step reachability on top of it to capture the geometric abstraction of the transition graph and allow multi-way probabilistic inference. We then showed how to embed the C-step abstraction of a Markov transition graph and encode the reachability into an asymmetric similarity function through conditional binary NCE. Based on this asymmetric similarity function, we developed a reference state conditioned distance measure, which enables the identification of geometrically salient states as subgoals. We demonstrated the quality of the learned representations and their effectiveness in subgoal discovery in the domain of gridworld.

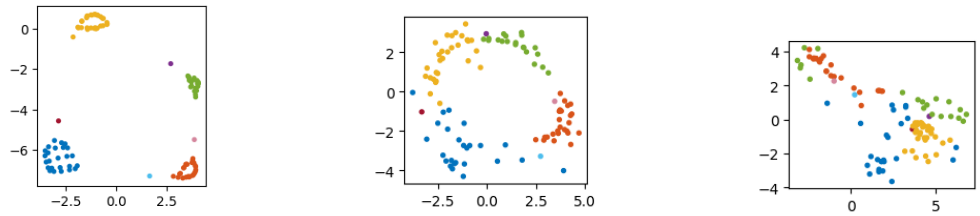
We leave the following topics for future work: (1) Extend the proposed method to learn the representations in continuous and high-dimensional state space. (2) Use the learned



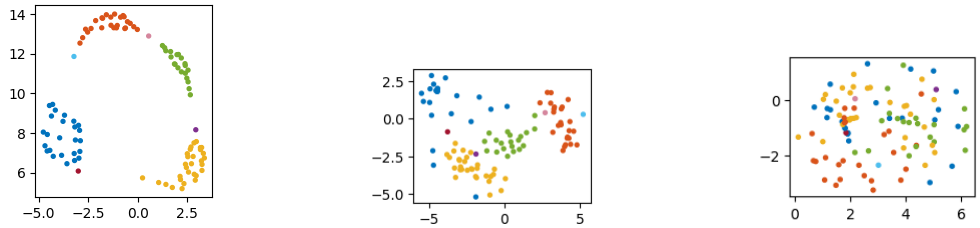
(a) $T = 153600$



(b) $T = 1024$

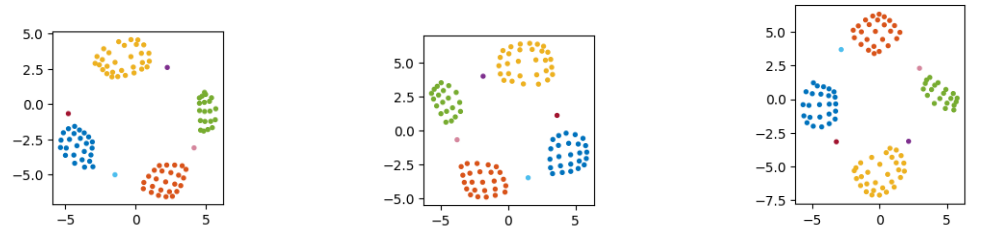


(c) $T = 300$

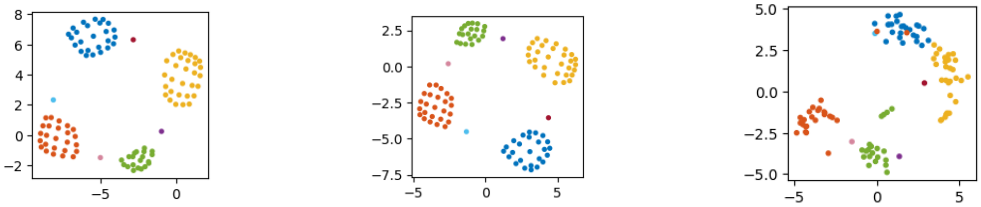


(d) $T = 150$

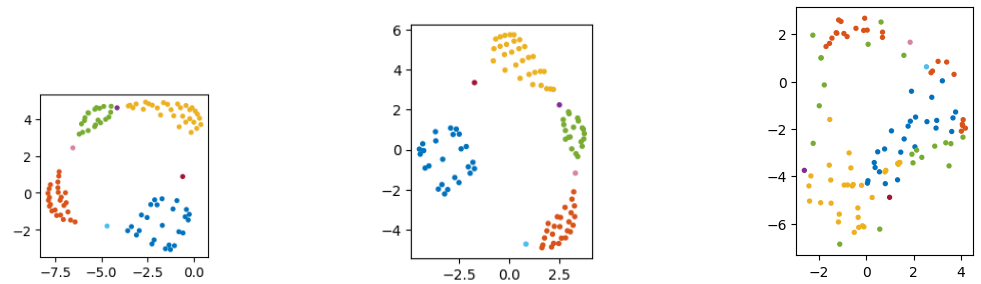
Figure 3.5. Learned representations with different negative distributions in Four Rooms environment when $C = 16$. Left column: $P_n(X) = P_X(X)$, Middle column: $P_n(X) = P_Y(X)$, Right column: $P_n(X) = U(X)$. Each row corresponds to the results with a different episode length T. All experiments have the same number of training environment steps 153600.



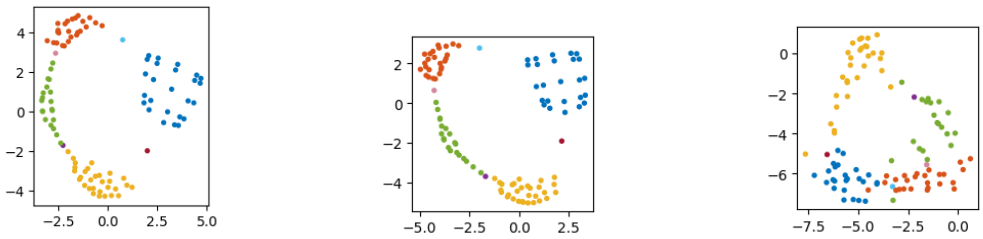
(a) $T = 153600$



(b) $T = 1024$



(c) $T = 300$



(d) $T = 150$

Figure 3.6. Learned representations with different negative distributions in Four Rooms environment when $C = 1$. Left column: $P_n(X) = P_X(X)$, Middle column: $P_n(X) = P_Y(X)$, Right column: $P_n(X) = U(X)$. Each row corresponds to the results with a different episode length T. All experiments have the same number of training environment steps 153600.

subgoals in hierarchical planning, reasoning, and HRL settings to solve long-horizon tasks. Unlike the model-free RL algorithms, our method effectively utilizes the data collected by random exploration to identify subgoals and build the environment model. Therefore, the sample efficiency is expected to be largely improved. (3) Use the learned similarity function as an intrinsic reward function to improve the performance of an RL agent. (4) Combine the proposed method of probabilistic directed graph embedding with the generative models to regularize the generated content and make them align well with how the world works.

Chapter 3, in part, is a reprint of the material as it appears in "Probabilistic World Modeling with Asymmetric Distance Measure", Meng Song, Geometry-grounded Representation Learning and Generative Modeling Workshop at International Conference on Machine Learning, 2024. The dissertation author was the primary investigator and author of this paper.

Appendix A

Learning Physics-Aware Rearrangement in Indoor Scenes

A.1 Simulated Environments and Robot

We provide the basic setup of our simulated environment and robot in Table A.1 and the task specifications in Table A.2,A.3,A.4.

A.2 Training Details

We use the PPO implementation of RLlib [111] without any modifications. All of our agents are trained using one GeForce RTX 2080 Ti GPU and 10 Intel(R) Core(TM) i9-10900K CPUs. To train a PPO agent, we distribute 10 rollout workers on CPUs to collect data in parallel. Training one agent for one million environment time steps takes approximately one hour.

A.2.1 Neural Network Architectures

The policy network and value network each consists of two fully-connected layers with 512 hidden units per layer and tanh activation, followed by a 512-unit linear output layer. The policy network and value network have no shared layers.

Table A.1. Basic parameters for the environments and robot.

Parameter	**Value**
Gravity (m/s^2)	(0, 0, -9.8)
Room dimension (m)	(6.95, 5.53, 3)
Layout range (m)	x=[-3.42, 3.53], y=[-2.80, 2.72], z=[0,3]
Robot platform	Fetch
Robot mass (kg)	50
Robot maximum wheel angular velocity (rad/s)	8.7
Robot dimension (m)	(0.51, 0.56, 1.1)
Physics simulation time step (s)	1/240
Rendering and action time step (s)	0.1
Maximum (action) time steps per episode	400

Table A.2. Experimental setup for variable mass pushing task.

Parameter	**Value**
Box dimension (m)	(0.45, 0.97, 0.72)
Box1 mass (kg)	50
Box2 mass (kg)	10
Box1 material	Steel oxydized bright
Box2 material	Wood hemlock
Box1 lateral friction coefficient	0.5
Box2 lateral friction coefficient	0.5
Floor lateral friction coefficient	0.5
Robot initial x-y position (m)	(0, 0)
Circular goal region center (m)	(0, 0)
Circular goal region radius (m)	1.5

Table A.3. Configuration setup for variable mass pushing task.

	Four facing directions			
Robot initial orientation around z axis (rad)	0	$-\pi/2$	$-\pi$	$-3\pi/2$
Left box initial x-y position (m)	(0.7, -0.7)	(0.7, 0.7)	(-0.7, 0.7)	(-0.7, -0.7)
Right box initial x-y position (m)	(0.7, 0.7)	(-0.7, 0.7)	(-0.7, -0.7)	(0.7, -0.7)

A.2.2 Hyperparameters

Table A.5 lists the common PPO hyperparameters used in the experiments of two tasks. Table A.6 lists the hyperparameters tuned for each task. In the experiments of the variable friction pushing task, we found that the KL coefficient and clip parameter are sensitive to the choice of random seeds and different reward settings. We list the tuned values in Table A.7.

Table A.4. Experimental setup for variable friction pushing task.

Parameter	Value
Box mass (kg)	10
Box dimension (m)	(0.45, 0.45, 0.72)
Low-friction floor material	Sls alumide polished rosy red
High-friction floor material	Carpet loop
Low-friction floor lateral friction coefficient	0.2
High-friction floor lateral friction coefficient	0.8
Box lateral friction coefficient	0.5
Band boundary (m)	y=0
Robot initial x-y position (m)	(-1.8, -0.4)
Box initial x-y position (m)	(-1.2, -0.4)
Robot initial orientation around z axis (rad)	0
Box initial orientation around z axis (rad)	0
Rectangular goal region range (m)	x=[1.2, 3.53], y=[0.4, 2.72]

Table A.5. PPO common hyperparameters.

Parameter	Value
optimizer	Adam
learning rate	10^{-4}
discount factor (γ)	0.99
GAE (λ)	0.98
KL target	0.01
entropy coefficient	0
value function clip parameter	10.0
steps per epoch (train batch size)	4000
SGD mini-batch size	512

Table A.6. PPO task specific hyperparameters.

Task	KL coefficient	Clip parameter
Variable Friction Pushing Task	0.3, 0.2	0.34, 0.35
Variable Mass Pushing Task	0.3	0.3

A.3 More Detailed Results

A.3.1 Variable Friction Pushing Task

Figure 3 in the main paper compares the training curves of two agents averaged over three random seeds. Here we further provide the training curve evaluated on the successful

Table A.7. PPO hyperparameters for variable friction pushing task.

Seed	Energy	KL coefficient	Clip parameter
1	w	0.3	0.34
1	w/o	0.3	0.34
4	w	0.3	0.35
4	w/o	0.3	0.35
8	w	0.2	0.35
8	w/o	0.3	0.35

(a) Seed 1 **(b)** Seed 4 **(c)** Seed 8

Figure A.1. Training curves of PPO under each of three random seeds for variable friction pushing tasks. Red: baseline policy trained without energy rewards. Blue: energy-aware policy trained with energy rewards.

episode energy cost under each of these three random seeds in Figure A.1.

A.3.2 Variable Mass Pushing Task

The training curves evaluated on the successful episode energy cost for each of the 10 configurations are provided in Figure A.2.

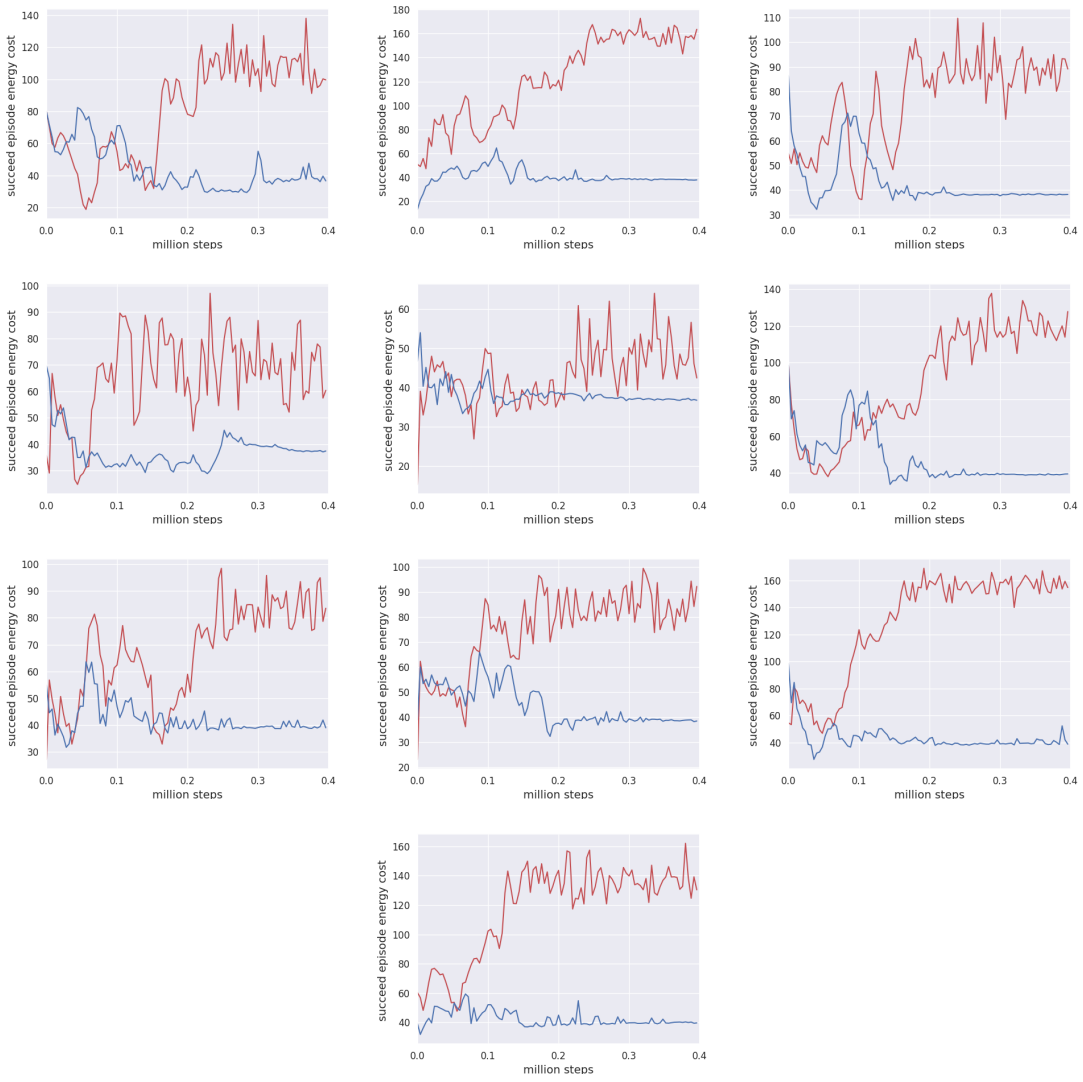


Figure A.2. Training curves of PPO for each of 10 configurations in variable mass pushing task. Red: baseline policy trained without energy rewards. Blue: energy-aware policy trained with energy rewards.

Appendix B

A Minimalist Prompt for Zero Shot Policy Learning

B.1 Additional Experiments and Ablations

B.1.1 Additional experiments on D4RL Maze2D

In addition to control and manipulation tasks, we investigate whether minimalist prompting DT can also perform well in navigation tasks. We experiment with Maze2D environment from the offline RL benchmark D4RL [75]. The Maze2D domain is a continuous 2D navigation task that requires a point mass to navigate to a goal location. We use the medium maze layout and create 21 training tasks and 5 test tasks. The tasks differ in the goal locations while the initial locations in each task are selected randomly for both training and evaluation episodes. The goal locations of the test tasks are unseen during the training. The agent has a 4-dimensional state space which includes its current 2D position and 2D velocity. The reward function $r_g(s,a)$ of each task is parameterized by the corresponding goal location g. Note that the goal location is not observable to the agent.

We only use the goal location as the task parameter but not the initial location which is akin to common real-world scenarios where a human is asking a robot to navigate to a goal location and the robot is expected to know its starting location without explicit instructions. This setup poses more challenging generalization requirements during evaluation where the agent

Table B.1. Comparison on Maze2D

Problem	Task-Learned-DT (Ours)	Task-DT (Ours)	Trajectory-DT	Pure-Learned-DT	DT
Maze2D	**23.00 ± 2.56**	17.58 ± 2.49	18.34 ± 2.89	2.04 ± 0.37	2.21 ± 1.30

should not only generalize to unseen goal locations but also unseen initial locations.

We compare Task-Learned-DT with four baselines in terms of zero-shot generalization in Table B.1. It is observed that Task-Learned-DT still outperforms Trajectory-DT by a large margin. All methods are trained on the expert demonstration trajectories generated by running the provided expert controller [75].

B.1.2 Comparison on medium and random data

To study the zero-shot generalization performance of Task-Learned-DT when expert demonstrations are not available, we train Task-Learned-DT, Task-DT, and Trajectory-DT on the medium and random trajectories on Cheetah-Velocity (Table B.2). Trajectory-DT is conditioned on the trajectory segments of the same quality in both training and test tasks. Although all of these methods experience a significant performance drop, our methods still perform better than Trajectory-DT. In particular, Task-Learned-DT outperforms Task-DT on medium and random data. This suggests that the learned prompt provides some robustness against the noises in the data. Based on the theoretical analysis of prior work [79], the performance drop of DT-based methods on sub-optimal data is possibly attributed to the limitations of behavior cloning.

B.1.3 Does minimalist-prompting DT perform well in sparse reward settings?

To investigate whether minimalist-prompting DT relies on the densely populated rewards, we create a delayed return version of ML1-Pick-Place, Ant-Direction, and Cheetah-Velocity following the original DT paper [60]. The zero-shot generalization performance is evaluated in Table B.3. We observed that both Task-Learned-DT and Task-DT are minimally affected by the sparsity of the rewards in most cases.

Table B.2. Ablating different data qualities on Cheetah-Velocity

Quality	Task-Learned-DT (Ours)	Task-DT (Ours)	Trajectory-DT
Expert	88.43 ± 3.15	**92.29 ± 3.81**	91.64 ± 1.83
Medium	**20.04 ± 0.16**	19.30 ± 1.74	16.58 ± 3.72
Random	**21.95 ± 4.22**	17.41 ± 8.46	19.86 ± 3.47

Table B.3. Ablation on sparse rewards

Problem	Task-Learned-DT (Ours)		Task-DT (Ours)	
	Original (Dense)	Delayed (Sparse)	Original (Dense)	Delayed (Sparse)
ML1-Pick-Place	92.52 ± 3.03	93.58 ± 3.24	92.58 ± 1.99	92.20 ± 3.44
Ant-Direction	110.35 ± 9.10	102.15 ± 1.92	89.96 ± 4.41	88.92 ± 2.09
Cheetah-Velocity	88.43 ± 3.15	89.63 ± 1.28	92.29 ± 3.81	91.41 ± 3.70

Table B.4. Seen task performance on five benchmark problems

Problem	Task-Learned-DT (Ours)	Task-DT (Ours)	Trajectory-DT	Pure-Learned-DT	DT
ML1-Pick-Place	98.41 ± 2.36	99.63 ± 0.32	**100.28 ± 0.08**	71.27 ± 1.09	76.94 ± 1.02
ML1-Reach	99.76 ± 0.23	**99.81 ± 0.06**	98.24 ± 2.68	72.13 ± 5.87	62.69 ± 3.92
ML1-Push	**142.97 ± 6.77**	133.30 ± 7.41	136.65 ± 3.06	99.57 ± 6.35	101.94 ± 3.48
Ant-Direction	89.57 ± 2.48	84.96 ± 2.05	**92.76 ± 0.10**	26.36 ± 1.48	29.53 ± 1.21
Cheetah-Velocity	96.88 ± 0.60	**97.53 ± 0.53**	96.41 ± 1.21	48.70 ± 1.10	47.19 ± 3.68

B.1.4 How does minimalist-prompting DT perform on the seen tasks?

We evaluate Task-Learned-DT and four baselines on the training tasks of five benchmark problems and report the performance in terms of mean normalized score in Table B.4 and Figure B.1. In ML1-Reach, ML1-Push, and Cheetah-Velocity, Task-Learned-DT (Task-DT) outperforms Trajectory-DT and maintains high performance in the rest two tasks. This observation shows that Task-Learned-DT (Task-DT) is able to improve zero-shot generalization without harming performance much on the seen tasks.

B.2 Task and Dataset Details

B.2.1 Task parameters

To help understand our approach, we provide concrete examples of the task parameters we adopted in each problem in Table B.5. The task parameters we experiment with capture the

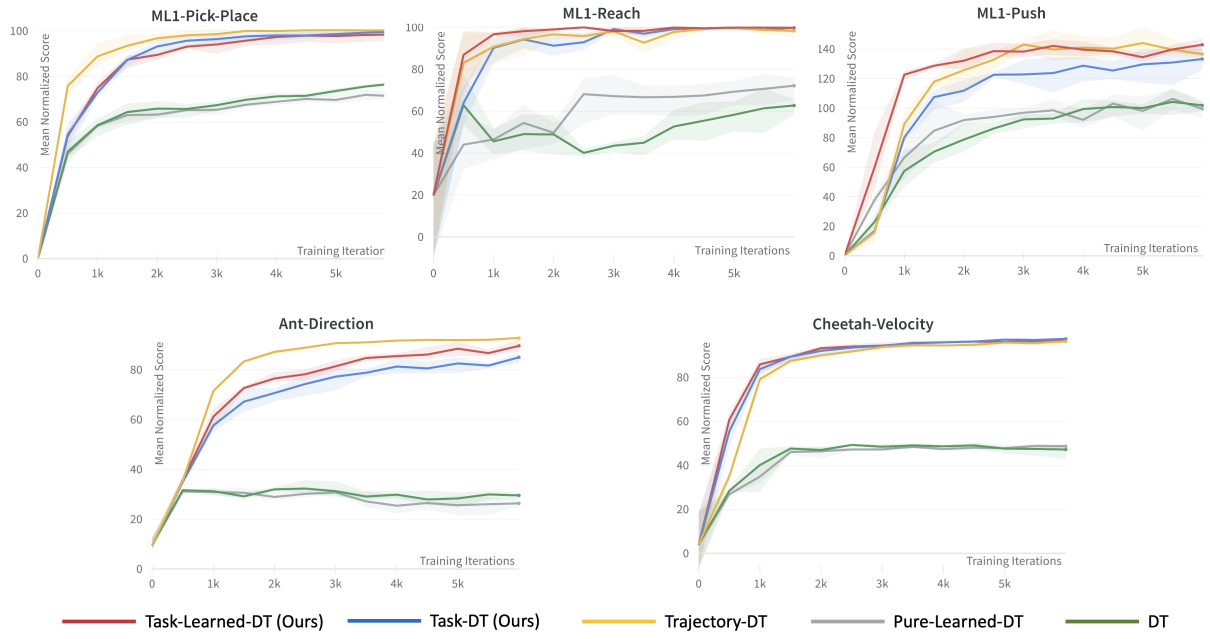


Figure B.1. Comparing full minimalist-prompting DT (Task-Learned-DT) with four baselines Task-DT, Trajectory-DT, Pure-Learned-DT and DT on training tasks. The seen task performance of each algorithm is evaluated through the entire training process on five benchmark problems and reported under the mean normalized score. Each training task is evaluated for 20 episodes. Shaded regions show one standard deviation of three seeds.

Table B.5. Examples of task parameters

Problem	Task Parameter	
	Problem-specific Meaning	Example
Cheetah-Velocity	goal velocity	[0.08]
Ant-Direction	goal direction	[1.20]
ML1-Pick-Place	target position	[0.01, 0.84, 0.22]
ML1-Reach	target position	[-0.07, 0.81, 0.08]
ML1-Push	target position	[0.02, 0.84, 0.02]
ML10	target position	[-0.30, 0.53, 0.13]
Maze2D	goal location	[1,1]

minimally sufficient task variations.

B.2.2 Task splits

We summarize the splits of training and test tasks for each problem in Table B.6. For Cheetah-Velocity and Ant-Direction, we follow the splits provided by [43]. For the rest problems,

Table B.6. Task splits

Problem	Number of training tasks	Number of test tasks
Cheetah-Velocity	35	5
Ant-Direction	45	5
ML1-Pick-Place	45	5
ML1-Reach	45	5
ML1-Push	45	5
ML10	450 (45 for each meta-task)	50 (5 for each meta-task)
Maze2D	21	5

we randomly sample a subset of tasks for evaluation.

In particular, we list the goal locations for training and test tasks in Maze2D. Maze2D with the medium layout is a 6×6 maze. We choose goal locations as the integer coordinates of all 26 empty cells in the maze and split them into training and test sets as follows:

The list of 21 training tasks:

- Task indices: 0, 2, 3, 4, 5, 7, 8, 9, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
- Goal locations: [1,1], [1,5], [1,6], [2,1], [2,2], [2,5], [2,6], [3,2], [3,4], [4,2], [4,4], [4,5], [4,6], [5,1], [5,3], [5,4], [5,6], [6,1], [6,2], [6,3], [6,5]

The list of 5 test tasks:

- Task indices: 1, 6, 10, 12, 25
- Goal locations: [1,2], [2,4], [3,3], [4,1], [6,6]

B.2.3 Data generation

For Cheetah-Velocity and Ant-Direction, we use the expert data released by [43] for training and evaluation. The medium and random data are generated from the replay buffer transitions provided by [48] and follow the same instructions. For ML1 and ML10, the expert trajectories for each task are generated by running the expert policy provided by Meta-World

Table B.7. Number of trajectories per task

Problem	Number of trajectories per task	Number of prompt trajectories per task
Cheetah-Velocity	999	5
Ant-Direction	1000	5
ML1-Pick-Place	100	5
ML1-Reach	100	5
ML1-Push	100	5
ML10	100	5
Maze2D	100	5

Table B.8. ML10 Meta-tasks

Meta-task	Description
reach-v2	Reach a goal position. Randomize the goal positions.
push-v2	Push the puck to a goal. Randomize puck and goal positions.
pick-place-v2	Pick and place a puck to a goal. Randomize puck and goal positions.
door-open-v2	Open a door with a revolving joint. Randomize door positions.
drawer-close-v2	Push and close a drawer. Randomize the drawer positions.
button-press-topdown-v2	Press a button from the top. Randomize button positions.
peg-insert-side-v2	Insert a peg sideways. Randomize peg and goal positions.
window-open-v2	Push and open a window. Randomize window positions.
sweep-v2	Sweep a puck off the table. Randomize puck positions.
basketball-v2	Dunk the basketball into the basket. Randomize basketball and basket positions.

[77]. For Maze2D, the expert trajectories for each task are generated by running the expert controller provided by D4RL [75].

We collect a set of trajectories for each training and test task and randomly sample a subset for Trajectory-DT to extract trajectory prompts. Note that the trajectories for the test tasks are only used by Trajectory-DT during the evaluation. We list the number of demonstration trajectories collected per task for each problem in Table B.7.

B.2.4 ML10 meta-tasks

To show the diversity of the skills we train the agent to master in ML10, we include a description of each of the 10 meta-tasks in Table B.8. Please refer to [77] for more details.

Table B.9. Common hyperparameters

Hyperparameter	Value
Context length K	20
Number of layers	3
Number of attention heads	1
Embedding dimension	128
Nonlinearity	ReLU
Dropout	0.1
Learning rate	10^{-3} when learned prompt length = 30 10^{-4} otherwise
Weight decay	10^{-4}
Warmup steps	10^{4}
Training batch size for each task	16
Number of evaluation episodes for each task	20
Number of gradient steps per iteration	10
Maximum training iterations	15500 ML10 6000 otherwise
Random seeds	1,6,8

B.3 Hyperparameters and Implementation Details

B.3.1 Hyperparameters

We describe the hyperparameters shared by Task-Learned-DT, Task-DT, Trajectory-DT, Pure-Learned-DT, and DT in Table B.9. For a fair comparison, we set most of the hyperparameter values to be the same as [43]. The evaluation results are reported when the maximum training iterations are reached and the training converged. All of the methods are run across the same three random seeds 1,6,8.

We show the hyperparameters specific to our method and Trajectory-DT in Table B.10 and Table B.11 respectively. Trajectory-DT assembles trajectory segments of length H from J episodes as a trajectory prompt. We set the values of H and J as the ones reported in [43] in Cheetah-Velocity, Ant-Direction, and ML1-Reach. The rest problems use the same values as ML1-Reach.

Table B.10. Problem-specific hyperparameters of Task-Learned-DT and Pure-Learned-DT

Problem	Learned Prompt Length
Cheetah-Velocity	15
Ant-Direction	15
ML1-Pick-Place	15
ML1-Reach	30
ML1-Push	30
ML10	30
Maze2D	30

Table B.11. Problem-specific hyperparameters of Trajectory-DT

Problem	The Number of Episodes J	Trajectory Segment Length H
Cheetah-Velocity	1	5
Ant-Direction	1	5
ML1-Pick-Place	1	2
ML1-Reach	1	2
ML1-Push	1	2
ML10	1	2
Maze2D	1	2

B.3.2 Target returns-to-go

In the experiments, we set the target returns-to-go for each task as the highest return of the associated expert demonstration trajectories. This is possible because we train with expert trajectories and Trajectory-DT requires expert trajectories even in the test tasks. However, setting the target returns-to-go according to the expert performance is not necessary. We can also set it based on our prior knowledge about the task, e.g. set it as the highest possible return inferred from the upper bound of the reward function, or set it as the return representing a satisfactory performance.

B.3.3 Implementation

We implement our method based on the released repository of [43].

B.3.4 Training

Our model is trained using NVIDIA A100 Tensor Core GPUs. Training our model using a single A100 GPU on a benchmark problem takes about 3 hours for typically 6000 training iterations. Evaluating the model in training and test tasks every 500 iterations throughout the training process takes another 3 hours due to the online interactions with the simulated environments.

B.4 Failure modes

By visualizing the trained policy performing the test tasks, we have observed that most failures occur when the agent has difficulty in precisely achieving unseen goal locations or directions, rather than struggling with mastering basic skills such as running or holding the block. This implies that including the task variations in the prompt or having the capability to extract task variations from the prompt is important to guarantee the task's success.

Appendix C

Probabilistic World Modeling with Asymmetric Distance Measure

C.1 Proof of C-step Approximation

Given a data set of N trajectories drawn from a T-step Markov chain M, i.e. $\mathscr{T} = \{(\mathbf{s}_0^i, \mathbf{s}_1^i, \ldots, \mathbf{s}_T^i)\}_{i=1}^N \sim \mathscr{M}$. Let Y represent a preceding random variable in the chain $Y = \{S_i \mid 0 \leq i \leq T-1\}$ and X represent a random variable subsequent to Y within C time steps. $X = \{S_j \mid i < j \leq \min(i+C, T)\}$. P denotes the one-step transition matrix. We now derive $P(X|Y)$ in terms of P as follows:

Based on the occurrences of the ordered pair $(\mathbf{y}, \mathbf{x})$, we can derive the joint distribution of X and Y as

$$P(X = \mathbf{x}, Y = \mathbf{y}) = \frac{\sum_{t=0}^{T-C} \sum_{i=t+1}^{t+c} n(S_i = \mathbf{x}, S_t = \mathbf{y}) + \sum_{t=T-C+1}^{T-1} \sum_{i=t+1}^{T} n(S_i = \mathbf{x}, S_t = \mathbf{y})}{N(T-C+1)C + N\sum_{t=1}^{C-1} t} \quad \text{(C.1)}$$

where $n(S_i = \mathbf{x}, S_t = \mathbf{y})$ denotes the number of times $S_i = \mathbf{x}$ and $S_t = \mathbf{y}$ occur in data set $\mathscr{T}$.

Similarly, by counting the occurrences of each specific state $\mathbf{y}$, the marginal distribution of Y can be derived as

$$P_Y(Y = \mathbf{y}) = \frac{C\sum_{t=0}^{T-C} n(S_t = \mathbf{y}) + \sum_{t=T-C+1}^{T-1} (T-t) n(S_t = \mathbf{y})}{N(T-C+1)C + N\sum_{t=1}^{C-1} t} \quad \text{(C.2)}$$

where $n(S_t = \mathbf{y})$ denotes the number of times $S_t = \mathbf{y}$ occurs in data set $\mathscr{T}$.

By the definition of the Markov chain, we have

$$n(S_i = \mathbf{x}, S_t = \mathbf{y}) = n(S_t = \mathbf{y})(P^{i-t})_{\mathbf{yx}} \quad (i = t+1, t+2, \ldots) \tag{C.3}$$

Plugging (C.3) into (C.1), we have

$$P(X = \mathbf{x}, Y = \mathbf{y}) = \frac{\sum_{t=0}^{T-C} n(S_t = \mathbf{y}) \sum_{i=1}^{C} (P^i)_{\mathbf{yx}} + \sum_{t=T-C+1}^{T-1} \sum_{i=t+1}^{T} n(S_t = \mathbf{y})(P^{i-t})_{\mathbf{yx}}}{N(T-C+1)C + N\sum_{t=1}^{C-1} t} \tag{C.4}$$

Given (C.4) and (C.2), we can derive the conditional distribution as

$$\begin{aligned} P(X = \mathbf{x} \mid Y = \mathbf{y}) &= \frac{P(X = \mathbf{x}, Y = \mathbf{y})}{P(Y = \mathbf{y})} \\ &= \frac{\sum_{t=0}^{T-C} n(S_t = \mathbf{y}) \sum_{i=1}^{C} (P^i)_{\mathbf{yx}} + \sum_{t=T-C+1}^{T-1} \sum_{i=t+1}^{T} n(S_t = \mathbf{y})(P^{i-t})_{\mathbf{yx}}}{C\sum_{t=0}^{T-C} n(S_t = \mathbf{y}) + \sum_{t=T-C+1}^{T-1} (T-t) n(S_t = \mathbf{y})} \end{aligned} \tag{C.5}$$

When $0 < C \ll T$, we can further drop the second term of the nominator and denominator, which is equivalent to changing the range of Y to $Y = \{S_i \mid 0 \leq i \leq T-C\}$. That is,

$$\begin{aligned} P(X = \mathbf{x} \mid Y = \mathbf{y}) &\approx \frac{\sum_{t=0}^{T-C} n(S_t = \mathbf{y}) \sum_{i=1}^{C} (P^i)_{\mathbf{yx}}}{C\sum_{t=0}^{T-C} n(S_t = \mathbf{y})} \\ &= \frac{1}{C} \sum_{t=1}^{C} (P^t)_{\mathbf{yx}} \end{aligned} \tag{C.6}$$

C.2 Proof of Bayes Optimal Scoring Function

Suppose that we are training a binary classifier $P(C|X, Y = \mathbf{y}; \theta)$ to discriminate between positive data distribution $P(X|Y = \mathbf{y})$ and negative data distribution $P_n(X)$, $\forall \mathbf{y}$. The ratio of negative and positive samples is K. The Bayes optimal classifier $P(C|X, Y = \mathbf{y}; \theta)$ is a maximum

a posteriori (MAP) hypothesis satisfying

$$\begin{aligned}\frac{P(X|Y=\mathbf{y})}{P_n(X)} &= \frac{P(X|Y=\mathbf{y},C=1)}{P(X|C=0)} \\ &= \frac{P(C=1|X,Y=\mathbf{y};\theta)}{P(C=0|X,Y=\mathbf{y};\theta)}\frac{P(C=0)}{P(C=1)} \\ &= \frac{P(C=1|X,Y=\mathbf{y};\theta)}{P(C=0|X,Y=\mathbf{y};\theta)}K \\ &= \frac{P(C=1|X,Y=\mathbf{y};\theta)}{1-P(C=1|X,Y=\mathbf{y};\theta)}K, \quad \forall \mathbf{y}\end{aligned} \tag{C.7}$$

$$P(C=1|X,Y=\mathbf{y};\theta) = \frac{P(X|Y=\mathbf{y})}{P(X|Y=\mathbf{y})+KP_n(X)}, \quad \forall \mathbf{y} \tag{C.8}$$

When the classifier is a logistic function, we have

$$P(C=1|X,Y=\mathbf{y};\theta) = \frac{1}{1+\exp(-f_\theta(X,Y=\mathbf{y}))}, \quad \forall \mathbf{y} \tag{C.9}$$

Plugging (C.9) into (C.8), we have

$$\exp(f_\theta(X,Y=\mathbf{y})) = \frac{P(X|Y=\mathbf{y})}{KP_n(X)}, \quad \forall \mathbf{y} \tag{C.10}$$

Bibliography

[1] Richard Sutton. The reward hypothesis. *URL: http://incompleteideas.net/rlai.cs. ualberta.ca/RLAI/rewardhypothesis.html*, 2004.

[2] Bokui Shen, Fei Xia, Chengshu Li, Roberto Martın-Martın, Linxi Fan, Guanzhi Wang, Shyamal Buch, Claudia D'Arpino, Sanjana Srivastava, Lyne P Tchapmi, Kent Vainio, Li Fei-Fei, and Silvio Savarese. iGibson: A simulation environment for interactive tasks in large realistic scenes. *arXiv preprint*, 2020.

[3] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai, 2019.

[4] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[5] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. SAPIEN: A simulated part-based interactive environment. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[6] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

[7] Chuang Gan, Siyuan Zhou, Jeremy Schwartz, Seth Alter, Abhishek Bhandwaldar, Dan Gutfreund, Daniel L. K. Yamins, James J. DiCarlo, Josh H. McDermott, Antonio Torralba, and Joshua B. Tenenbaum. The threedworld transport challenge: A visually guided task-and-motion planning benchmark for physically realistic embodied AI. *CoRR*, abs/2103.14025, 2021.

[8] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee.

Transporter networks: Rearranging the visual world for robotic manipulation, 2021.

[9] Dhruv Batra, Angel X. Chang, Sonia Chernova, Andrew J. Davison, Jia Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Mottaghi, Manolis Savva, and Hao Su. Rearrangement: A challenge for embodied AI, 2020.

[10] Anton Bakhtin, Laurens van der Maaten, Justin Johnson, Laura Gustafson, and Ross Girshick. Phyre: A new benchmark for physical reasoning, 2019.

[11] Tian Ye, Xiaolong Wang, James Davidson, and Abhinav Gupta. Interpretable intuitive physics model. In *Proceedings of (ECCV) European Conference on Computer Vision*, pages 89 – 105, September 2018.

[12] Kexin Yi*, Chuang Gan*, Yunzhu Li, Pushmeet Kohli, Jiajun Wu, Antonio Torralba, and Joshua B. Tenenbaum. Clevrer: Collision events for video representation and reasoning. In *International Conference on Learning Representations*, 2020.

[13] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. Learning to see physics via visual de-animation. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[14] Michael Janner, Sergey Levine, William T. Freeman, Joshua B. Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-centric models. In *International Conference on Learning Representations*, 2019.

[15] Haozhi Qi, Xiaolong Wang, Deepak Pathak, Yi Ma, and Jitendra Malik. Learning long-term visual dynamics with region proposal interaction networks. In *International Conference on Learning Representations*, 2021.

[16] Yiding Jiang, Shixiang Gu, Kevin Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. *CoRR*, abs/1906.07343, 2019.

[17] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2021.

[18] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. *CoRR*, abs/1609.05143, 2016.

[19] OpenAI. Robogym. https://github.com/openai/robogym, 2020.

[20] Andrii Zadaianchuk, Maximilian Seitzer, and Georg Martius. Self-supervised visual

reinforcement learning with object-centric representations. In *International Conference on Learning Representations*, 2021.

[21] Kelsey R. Allen, Kevin A. Smith, and Joshua B. Tenenbaum. The tools challenge: Rapid trial-and-error learning in physical problem solving. *CoRR*, abs/1907.09620, 2019.

[22] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics, 2016.

[23] Daniel Bear, Chaofei Fan, Damian Mrowca, Yunzhu Li, Seth Alter, Aran Nayebi, Jeremy Schwartz, Li F Fei-Fei, Jiajun Wu, Josh Tenenbaum, and Daniel L Yamins. Learning physical graph representations from visual scenes. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6027–6039. Curran Associates, Inc., 2020.

[24] Zhijian Liu, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Physical primitive decomposition. In *European Conference on Computer Vision (ECCV)*, 2018.

[25] Jiajun Wu, Ilker Yildirim, Joseph J Lim, Bill Freeman, and Josh Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[26] Zhenjia Xu, Jiajun Wu, Andy Zeng, Joshua B Tenenbaum, and Shuran Song. Densephysnet: Learning dense physical object representations via multi-step dynamic interactions. In *Robotics: Science and Systems (RSS)*, 2019.

[27] Devendra Singh Chaplot, Ruslan Salakhutdinov, Abhinav Gupta, and Saurabh Gupta. Neural topological slam for visual navigation. In *CVPR*, 2020.

[28] Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. In *In Neural Information Processing Systems*, 2020.

[29] Luca Weihs, Aniruddha Kembhavi, Kiana Ehsani, Sarah M Pratt, Winson Han, Alvaro Herrasti, Eric Kolve, Dustin Schwenk, Roozbeh Mottaghi, and Ali Farhadi. Learning generalizable visual representations via interactive gameplay. In *International Conference on Learning Representations*, 2021.

[30] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing*

Systems, volume 30. Curran Associates, Inc., 2017.

[31] Bullet real-time physics simulation, 2018.

[32] Zhengqin Li, Ting-Wei Yu, Shen Sang, Sarah Wang, Meng Song, Yuhan Liu, Yu-Ying Yeh, Rui Zhu, Nitesh Gundavarapu, Jia Shi, Sai Bi, Zexiang Xu, Hong-Xing Yu, Kalyan Sunkavalli, Milos Hasan, Ravi Ramamoorthi, and Manmohan Chandraker. OpenRooms: An end-to-end open framework for photorealistic indoor scene datasets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[33] Melonee Wise, Michael Ferguson, Derek King, Eric Diehr, and David Dymesich. Fetch & freight: Standard platforms for service robot applications. In *2016 International Joint Conference on Artificial Intelligence*, 2016.

[34] Jean Tarbouriech, Runlong Zhou, Simon S. Du, Matteo Pirotta, Michal Valko, and Alessandro Lazaric. Stochastic shortest path: Minimax, parameter-free and towards horizon-free regret, 2021.

[35] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[36] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[37] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

[38] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023.

[39] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model, 2023.

[40] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[41] Homanga Bharadhwaj, Abhinav Gupta, Shubham Tulsiani, and Vikash Kumar. Zero-shot robot manipulation from passive human videos. *arXiv preprint arXiv:2302.02011*, 2023.

[42] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A Efros, and Trevor Darrell. Zero-shot visual imitation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 2050–2053, 2018.

[43] Mengdi Xu, Yikang Shen, Shun Zhang, Yuchen Lu, Ding Zhao, Joshua Tenenbaum, and Chuang Gan. Prompting decision transformer for few-shot policy generalization. In *International Conference on Machine Learning*, pages 24631–24645. PMLR, 2022.

[44] Mengdi Xu, Yuchen Lu, Yikang Shen, Shun Zhang, Ding Zhao, and Chuang Gan. Hyper-decision transformer for efficient online policy adaptation, 2023.

[45] Sudeep Dasari and Abhinav Gupta. Transformers for one-shot visual imitation. In *Conference on Robot Learning*, pages 2071–2084. PMLR, 2021.

[46] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5331–5340. PMLR, 09–15 Jun 2019.

[47] Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. In *International Conference on Machine Learning*, pages 1480–1490. PMLR, 2017.

[48] Eric Mitchell, Rafael Rafailov, Xue Bin Peng, Sergey Levine, and Chelsea Finn. Offline meta-reinforcement learning with advantage weighting. In *International Conference on Machine Learning*, pages 7780–7791. PMLR, 2021.

[49] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. *Advances in neural information processing systems*, 30, 2017.

[50] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work?, 2022.

[51] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 991–1002. PMLR, 08–11 Nov 2022.

[52] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020.

[53] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4693–4700. IEEE, 2018.

[54] Renhao Wang, Jiayuan Mao, Joy Hsu, Hang Zhao, Jiajun Wu, and Yang Gao. Programmatically grounded, compositionally generalizable robotic manipulation. *arXiv preprint arXiv:2304.13826*, 2023.

[55] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320. PMLR, 2015.

[56] Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. *arXiv preprint arXiv:1912.06088*, 2019.

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

[58] Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. Learning to generalize across long-horizon tasks from human demonstrations. *arXiv preprint arXiv:2003.06085*, 2020.

[59] Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. *Advances in neural information processing systems*, 32, 2019.

[60] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[61] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one

big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.

[62] Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. Rvs: What is essential for offline rl via supervised learning? *arXiv preprint arXiv:2112.10751*, 2021.

[63] Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

[64] Aviral Kumar, Xue Bin Peng, and Sergey Levine. Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*, 2019.

[65] Kuang-Huei Lee, Ofir Nachum, Mengjiao Sherry Yang, Lisa Lee, Daniel Freeman, Sergio Guadarrama, Ian Fischer, Winnie Xu, Eric Jang, Henryk Michalewski, et al. Multi-game decision transformers. *Advances in Neural Information Processing Systems*, 35:27921–27936, 2022.

[66] Hiroki Furuta, Yutaka Matsuo, and Shixiang Shane Gu. Generalized decision transformer for offline hindsight information matching. *arXiv preprint arXiv:2111.10364*, 2021.

[67] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. In *International Conference on Machine Learning*, pages 27042–27059. PMLR, 2022.

[68] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[69] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online, August 2021. Association for Computational Linguistics.

[70] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

[71] Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

[72] Annie Xie, James Harrison, and Chelsea Finn. Deep reinforcement learning amidst lifelong non-stationarity. *arXiv preprint arXiv:2006.10701*, 2020.

[73] Qiang Zhang, Tete Xiao, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Learning cross-domain correspondence for control with dynamics cycle-consistency. *arXiv preprint*

arXiv:2012.09811, 2020.

[74] Donald Hejna, Lerrel Pinto, and Pieter Abbeel. Hierarchically decoupled imitation for morphological transfer. In *International Conference on Machine Learning*, pages 4159–4171. PMLR, 2020.

[75] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[76] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *OpenAI preprint*, 2018.

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

[78] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[79] Aviral Kumar, Joey Hong, Anikait Singh, and Sergey Levine. When should we prefer offline reinforcement learning over behavioral cloning? *arXiv preprint arXiv:2204.05618*, 2022.

[80] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[81] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[82] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[83] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[84] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[85] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[86] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[87] Lawrence K Saul and Sam T Roweis. An introduction to locally linear embedding. *unpublished. Available at: http://www. cs. toronto. edu/˜ roweis/lle/publications. html*, 2000.

[88] Joshua B Tenenbaum, Vin de Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[89] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. *CoRR*, abs/1906.05253, 2019.

[90] Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

[91] Sherry Yang, Jacob Walker, Jack Parker-Holder, Yilun Du, Jake Bruce, Andre Barreto, Pieter Abbeel, and Dale Schuurmans. Video as the new language for real-world decision making. *arXiv preprint arXiv:2402.17139*, 2024.

[92] Michael U Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of machine learning research*, 13(2), 2012.

[93] Zhuang Ma and Michael Collins. Noise contrastive estimation and negative sampling for conditional models: Consistency and statistical efficiency. *CoRR*, abs/1809.01812, 2018.

[94] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[95] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020.

[96] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning, 2020.

[97] Yubei Chen, Adrien Bardes, Zengyi Li, and Yann LeCun. Bag of image patch embedding behind the success of self-supervised learning, 2023.

[98] Dibya Ghosh, Abhishek Gupta, and Sergey Levine. Learning actionable representations with goal-conditioned policies. *arXiv preprint arXiv:1811.07819*, 2018.

[99] Yifan Wu, George Tucker, and Ofir Nachum. The laplacian in rl: Learning representations with efficient approximations, 2018.

[100] Anirudh Goyal, Riashat Islam, Daniel Strouse, Zafarali Ahmed, Matthew Botvinick, Hugo Larochelle, Yoshua Bengio, and Sergey Levine. Infobot: Transfer and exploration via the information bottleneck. *arXiv preprint arXiv:1901.10902*, 2019.

[101] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning, 2018.

[102] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.

[103] Michael U. Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13(null):307–361, feb 2012.

[104] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2019.

[105] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. CURL: Contrastive unsupervised representations for reinforcement learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5639–5650. PMLR, 13–18 Jul 2020.

[106] Benjamin Eysenbach, Vivek Myers, Sergey Levine, and Ruslan Salakhutdinov. Contrastive representations make planning easy. In *NeurIPS 2023 Workshop on Generalization in Planning*, 2023.

[107] Tianjun Zhang, Benjamin Eysenbach, Ruslan Salakhutdinov, Sergey Levine, and Joseph E. Gonzalez. C-planning: An automatic curriculum for learning goal-reaching tasks. *CoRR*, abs/2110.12080, 2021.

[108] Amy McGovern and Andrew G Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. *ICONIP 2008*, 2001.

[109] Yecheng Jason Ma, Jason Yan, Dinesh Jayaraman, and Osbert Bastani. How far i'll go: Offline goal-conditioned reinforcement learning via f-advantage regression. *arXiv preprint arXiv:2206.03023*, 2022.

[110] Dingsheng Deng. Dbscan clustering algorithm based on density. In *2020 7th international forum on electrical engineering and automation (IFEEA)*, pages 949–953. IEEE, 2020.

[111] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg,

Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning, 2018.