

UNIVERSITÄT
DES
SAARLANDES

LEARNING SAFER MULTIMODAL DIFFUSION POLICIES USING LATENT REPRESENTATIONS

Bachelor Thesis

by

Nils Waltner

Advisor:
Joshua Meyer

1. Referee:
Prof. Dr. Verena Wolf

2. Referee:
Prof. Dr. Philipp Slusallek

September 8, 2026

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne die Beteiligung dritter Personen verfasst habe, und dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderweitigen fremden Äußerungen entnommen wurden, sind als solche kenntlich gemacht. Ich erkläre mich damit einverstanden, dass die Arbeit mittels eines Plagiatsprogrammes überprüft wird. Mir ist bewusst, dass der Verstoß gegen diese Versicherung zum Nichtbestehen der Prüfung bis hin zum Verlust des Prüfungsanspruchs führen kann.

Declaration of Original Authorship

I hereby declare that I have written this thesis independently and without the involvement of third parties, and that I have used no sources or aids other than those indicated. All passages taken directly or indirectly from publications or other external sources have been identified as such. I agree that the thesis may be checked using plagiarism detection software. I am aware that any violation of this declaration may result in failing the examination and lead to losing the right to be examined.

Saarbrücken, 11.09.26 Walther

Ort/Place, Datum/Date Unterschrift/Signature

Abstract

DDiffPG is a reinforcement learning algorithm that trains an agent to learn multiple ways to solve a given task using diffusion policies. In this thesis, we investigate an extension of DDiffPG that aims to make the algorithm more domain-independent through a latent-based clustering method. To achieve this, we implement an embedding network that translates state observations into a general latent space before applying the distance measure to the resulting trajectories. The proposed extension is evaluated in the Safety Gymnasium environments based on its ability to identify and preserve different behavior modes. The results show that the extended algorithm achieves performance comparable to the original DDiffPG algorithm on the Safety Gymnasium benchmark while providing greater flexibility in the representation and clustering of different state spaces. Additionally, we compare different embedding networks (ICM, EMI, CTEC) and find that CTEC, which focuses on preserving temporal trajectory structure, is best suited for encoding the spatial information required for trajectory clustering.

Acknowledgments

I would like to thank my advisor, Joshua Meyer, for suggesting this interesting topic and for his guidance throughout the writing of this thesis, including many helpful discussions and patient answers to my questions. I am also grateful to Timo P. Gros and Prof. Dr. Verena Wolf for welcoming me into their team and giving me the opportunity to write this thesis. My thanks also go to Prof. Dr. Philipp Slusallek for agreeing to serve as the second reviewer.

Finally, I would like to thank my family for their unwavering support throughout my studies.

Contents

I. Introduction **1**
I.1. Motivation . . . 1
I.2. Thesis Structure . . . 2

II. Background and Related Work **3**
II.1. Reinforcement Learning . . . 3
II.2. Latent Representation Learning . . . 4
II.2.1. ICM . . . 5
II.2.2. EMI . . . 5
II.2.3. CTEC . . . 6
II.3. Dynamic Time Warping . . . 6
II.4. Diffusion . . . 9
II.4.1. Diffusion Process . . . 9
II.4.2. Diffusion Policies . . . 10
II.5. Diffusion-QL . . . 10
II.6. DIPO . . . 11
II.7. DDiffPG . . . 12
II.8. Safety Gymnasium . . . 13

III. Methodology **15**
III.1. Reproduction of Original DDiffPG Experiments . . . 15
III.2. Adapting DDiffPG to Safety Gymnasium . . . 15
III.3. Proposed Latent DDiffPG Method . . . 16
III.4. Offline and Online Embedding Training . . . 17
III.5. Cost Integration . . . 17

IV. Empirical Evaluation **19**
IV.1. Environment Setup . . . 19
IV.2. Hyperparameters and Training Setup . . . 20
IV.3. Standard DDiffPG Results . . . 23
IV.3.1. Reproduction of DDiffPG Results . . . 23
IV.3.2. Standard DDiffPG on Safety Gymnasium . . . 23

IV.4. Latent DDiffPG Results 25
IV.4.1. Offline Latent DDiffPG 25
IV.4.2. Online Latent DDiffPG on AntMaze 28
IV.4.3. Online Latent DDiffPG on Safety Gymnasium 30
IV.5. Cost-aware Latent DDiffPG Results 35
IV.6. Discussion 37

V. Limitations, Future Work & Conclusion **39**
V.1. Limitations 39
V.1.1. Clustering Threshold 39
V.1.2. Embedding Network 39
V.1.3. Computational Cost of DTW 40
V.2. Future Work 40
V.2.1. Cost Implementation and Clustering Threshold 40
V.2.2. Trajectory Clustering 41
V.3. Conclusion 41

Bibliography **43**

A. Additional Experiment Results **45**
A.1. Behavior Modes Discovered by DDiffPG on Safety Gymnasium Pillars 45
A.2. Behavior Modes Discovered by Latent DDiffPG on Safety Gymnasium Pillars 47
A.3. Behavior Modes Discovered by Latent DDiffPG on Safety Gymnasium Wall 50

Chapter I.

Introduction

1. Motivation

In recent years, reinforcement learning (RL) has made many advances. Agents have improved in solving a wide variety of tasks, such as games including Go or chess (Silver et al., 2018), robotic control tasks (Kober et al., 2013), and autonomous driving (Kiran et al., 2021). Researchers have proposed many methods for improving an agent's performance by first exploring the environment and later exploiting the gained knowledge. These methods include entropy regularization (Haarnoja et al., 2018; Mnih et al., 2016), which encourages stochastic policies to promote exploration, and intrinsic motivation (Pathak et al., 2017; Burda et al., 2018), where internally generated rewards encourage the agent to seek novel states. However, many of those methods suffer from one limitation: *policies collapsing into a single mode*. Here, a mode refers to one specific behavior pattern followed by the agent while neglecting possible alternative trajectories. Executing only a fixed set of actions limits the agent's ability to reliably solve a given task. The agent's decision-making is mostly influenced and dominated by the received reward, steering it towards the path that yields the highest reward and dismissing other behavior modes that may generate less reward but solve the task just as reliably as the optimal mode. However, especially in domains like navigation or game playing, preserving multiple solutions or paths to a goal can be beneficial, even if they do not lead to the highest reward. Focusing on a single mode too early can hinder the agent from finding additional goals or even more optimal paths to the goal. In dynamic environments, this could completely stop the agent from reaching the goal after the environment dynamics have changed. Therefore, in such settings, we want our agent to continue exploring alternative solutions even after it has found what appears to be the optimal one.

Recent work introduces diffusion policies to tackle this issue (Yang et al., 2023; Li et al., 2024). Diffusion policies offer the agent the ability to learn multimodal

policies. Unlike standard Gaussian policies, which are unimodal, diffusion policies parameterize the action distribution through an iterative denoising process and can therefore represent arbitrary multimodal policies. An example of such an approach is the DDiffPG algorithm presented in the paper 'Learning Multimodal Behaviors from Scratch with Diffusion Policy Gradient' (Li et al., 2024). Although this approach seems promising, it relies on domain-specific information, which limits its applicability across different environments. The original DDiffPG algorithm can only be applied to a fixed set of environments, since transferring it to a new domain requires manually redefining the state clustering. The clustering itself is based on the 2D coordinates of the observed states and relies on an environment-specific threshold. Adapting these parameters could quickly become infeasible in domains where states are not as easily accessible or simply cannot be expressed as coordinates.

To overcome this limitation and make DDiffPG domain-independent, we propose extending DDiffPG with latent state clustering directly learned from the entire state observation. We evaluate our method extension against the standard DDiffPG algorithm in MuJoCo AntMaze and further apply it to the previously untested Safety Gymnasium benchmark (Ji et al., 2023), assessing performance, safety, and its ability to identify and preserve distinct behavior modes.

2. Thesis Structure

First, in Chapter II we introduce and explain the underlying concepts used in the original DDiffPG algorithm, together with three embedding networks from the intrinsic motivation literature that will be incorporated into the extension. We explain our planned approach for extending the algorithm in Chapter III. In Chapter IV we present the experimental setup and interpret and discuss the results obtained from the experiments. We conclude the thesis with a discussion of the limitations and an outlook on future work in Chapter V.

Chapter II.

Background and Related Work

1. Reinforcement Learning

RL studies how an agent can learn to make decisions to solve a task. The agent learns by interacting with the environment and receiving feedback in terms of a reward. Said environment is formalized by a discounted Markov decision process (MDP) $(S, A, r, p, \rho_0, \gamma)$. S is the state space and A the action space, $r : S \times A \times S \to \mathbb{R}$ is the reward function, $p : S \times A \times S \to \mathbb{R}^+$ is the transition function, $\rho_0 : S \to \mathbb{R}^+$ defines the initial states, and $\gamma \in [0; 1]$ is the discount factor. The agent's goal is to learn a policy $\pi : S \times A \to \mathbb{R}^+$ where $\pi(a|s)$ is the probability of choosing action a in state s. The performance of a policy is measured by its *return*, described as the cumulative discounted reward over time T, $G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$. Formally, the optimization objective is defined as

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r_t \mid s_0 \sim \rho_0 \right].$$

The objective is then to find an optimal policy

$$\pi^* = \arg\max_\pi J(\pi).$$

There exist multiple methods to solve this problem. For simple environments, the Bellman equations can be solved via dynamic programming to find the optimal policy. However, this quickly becomes infeasible in more complicated environments, creating the need for more involved methods. Value-based methods approximate the expected return in different states, such as Q-learning (Watkins and Dayan, 1992) or DQN (Mnih et al., 2015). Another class of methods is policy-based methods, which directly learn a policy that maps the state to the action, and update the policy using gradient-based methods. Example algorithms in this class include basic policy gradient methods like REINFORCE (Williams, 1992). A third major class is actor-critic methods which combine value- and policy-based approaches.

Examples include PPO (Schulman et al., 2017), SAC (Haarnoja et al., 2018), and A3C (Mnih et al., 2016).

Exploration Exploration plays an essential part in RL, as it measures the agent's ability to discover new, previously unseen states of the environment. There are different techniques which motivate the agent not only to follow the path of the highest seen return (exploitation), but also to encourage it to explore unseen areas. The trade-off between exploration and exploitation represents a major challenge in training a successful agent. Without exploration, the agent is most likely stuck in local minima by committing too early to a discovered strategy. It will not be able to find alternative modes or goals. However, focusing too much on exploration can prevent the agent from learning effectively, since it might spend all its time searching for novel states instead of exploiting what it has learned (Ladosz et al., 2022).

Safe Reinforcement Learning Exploration in general is rather unsafe. In order to explore the entire state space, the agent needs to execute actions with unknown consequences. When training in a simulated environment, any mistakes or unintended behaviors have no real-world consequences. In real-world settings, it is crucial to take precautions and lead the agent in the direction of safe trajectories, ensuring that it reaches the goal safely, rather than quickly. For instance, in a traffic control environment, multiple paths could lead to one goal: one may pass over an icy road, presenting the risk of crashing, while another is completely safe. During exploration, the agent should discover and learn to prefer the safe path rather than risk crashing on the faster one. Safe RL addresses this by extending the basic RL framework with safety constraints. For example, when trying to maximize the cumulative return, the agent must respect an underlying safety cost function that balances reward and risk (Kushwaha et al., 2025).

2. Latent Representation Learning

Latent representation learning aims to learn meaningful features from a given input by mapping the input data into a latent space. Typically, a neural network (NN) is trained to transform the input through a series of layers, learning representations that capture relevant characteristics of the data. Different learning approaches can lead to different latent spaces, as the chosen objective determines which types of information are emphasized and preserved. The resulting latent representation provides a compact representation of the input and can detect underlying patterns that may not be directly observable in the original input data. Depending on the

learning approach, latent representations can be used to reduce the dimensionality of the input while retaining information that is relevant to the task.

2.1. ICM

The intrinsic curiosity module (ICM) (Figure II.1) is introduced by Pathak et al. (2017). In their paper, they differentiate between three sources that can modify the agent's observation: (1) things that can be controlled by the agent, (2) things that the agent cannot control but that affect the agent, and (3) things out of the agent's control and not affecting the agent. An example of something that does not affect the agent (3) could be leaves on a tree. The agent can observe the movement of the tree leaves in a breeze. Since it is difficult to model a breeze, it is even harder to predict the location of each individual leaf. As a result, the prediction error remains high and the agent stays curious about the leaves, even though they do not affect it. The embedding network now aims to learn a state representation that models (1) and (2). To achieve this, it contains three different models: the feature encoder, the forward model, and the inverse model. The feature encoder later serves as our embedding module. It receives as input the state and returns its latent representation. The forward and inverse models are used to train the feature encoder. The forward model receives the action a_t and the latent representation of the current state $\phi(s_t)$ as input and tries to predict the latent representation of the following state $\hat{\phi}(s_{t+1})$. The inverse model receives the latent representations of the state pair (s_t, s_{t+1}) and tries to predict the action that led to this transition. The feature encoder is then trained on the basis of the weighted losses of the two models.

2.2. EMI

The paper by Kim et al. (2018) introduces an embedding network (Figure II.2), which learns two encoder functions mapping states and actions into a shared low-dimensional space. Rather than relying on reconstruction, the training objective is based on mutual information maximization: the state and action embeddings should be predictive of the corresponding next state embedding, and the state and next state embeddings should in turn be predictive of the action embedding. This avoids the need to encode full high-dimensional observations, making the approach better suited to complex, high-dimensional input spaces such as images. The goal is to capture the non-linear aspects of the dynamics in the encoders, such that the dynamics become linear in the embedding space. However, in such a linear dynamics model, some transitions cannot be explained linearly, resulting in irreducible error. To account for this, the authors introduce a separate error network that estimates this irreducible error. The final training objective combines

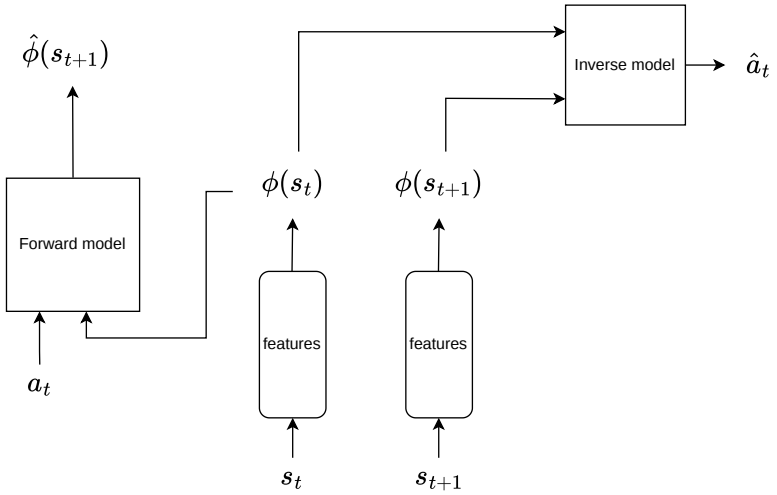


Figure II.1.: High-level overview of the ICM embedding process.

the mutual information loss, linear dynamics reconstruction loss, and an error regularization term.

2.3. CTEC

The embedding network we use as CTEC (Figure II.3) is presented in the paper by Mohamed et al. (2026), where it is referred to as C-TeC; we adopted the name CTEC for consistency with our notation. It focuses on preserving the temporal structure rather than reconstructing complete observations. The model learns a state-action encoder and a future state encoder, trained such that positive pairs (a state-action pair (s, a) and its future state s_f) have high similarity and negative pairs (mismatched (s, a) and s_f combinations) have low similarity. The future state is sampled from the states following the (s, a) transition within the same trajectory, using a geometric distribution that assigns higher probability to states closer in time. The encoders are trained using the InfoNCE contrastive loss. As a result, the embedding of a state-action pair is pulled close to the representation of the future state it actually leads to and pushed away from the representations of unrelated future states.

3. Dynamic Time Warping

Dynamic time warping (DTW) is an algorithm used to align and compare two time-series datasets. Its basic idea is introduced in a paper by Sakoe and Chiba (1978). The algorithm creates alignment by mapping individual points between

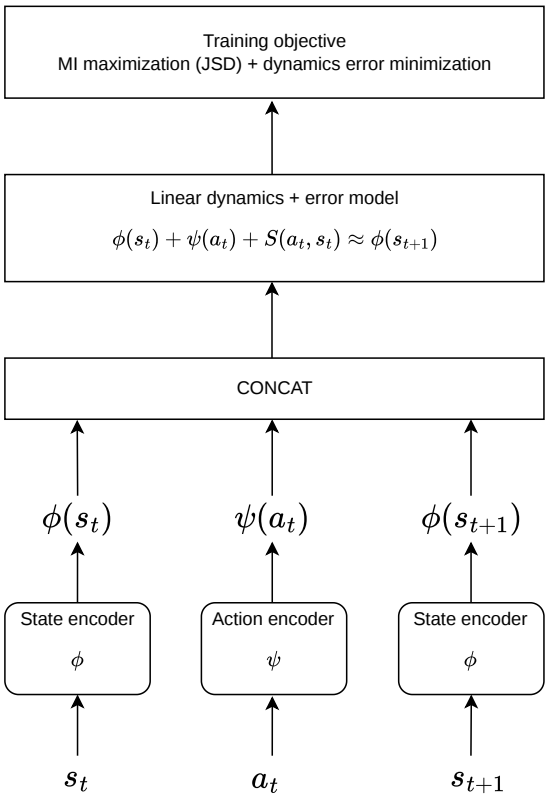


Figure II.2.: High-level overview of the EMI embedding process.

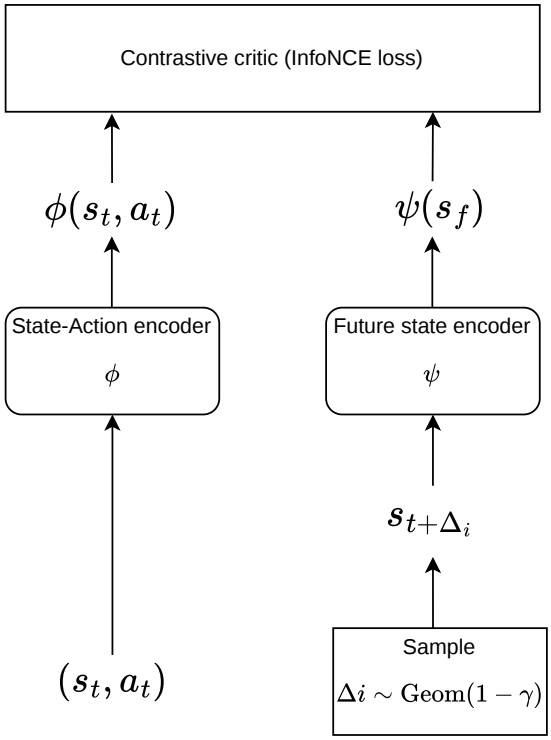


Figure II.3.: High-level overview of the CTEC embedding process.

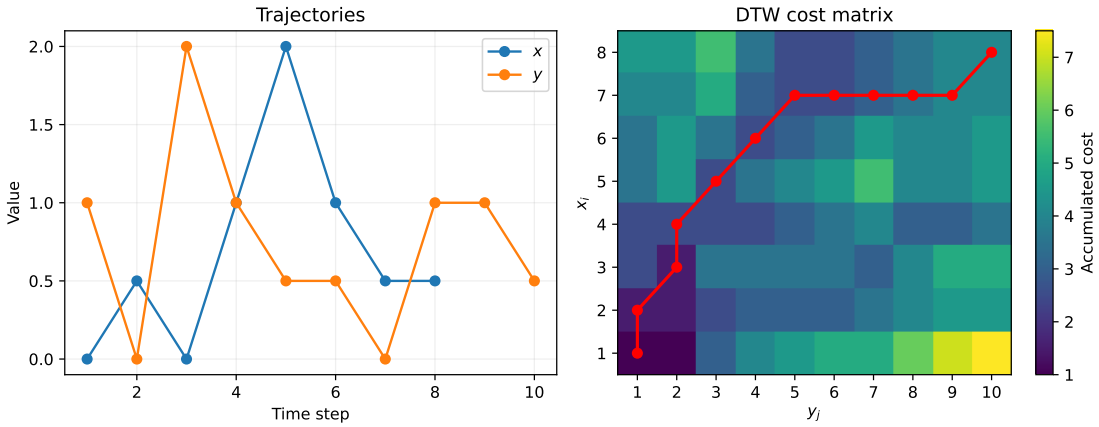


Figure II.4.: Distance calculation using dynamic time warping.

two series and calculating the distance between the corresponding points. This makes it possible to calculate the distance between two trajectories of unequal length. DTW receives two time series $x_{1:N}$ and $y_{1:M}$ as input, initializes a cost matrix D, and chooses a point-by-point distance measure, which is the squared Euclidean distance by default. It then traverses the cost matrix row by row and fills each entry with the calculated point-wise cost plus the accumulated cost of the lowest-cost neighboring cell. For each entry, the neighboring cell that contributes the lowest cost is stored to establish a path through the matrix. After filling the entire matrix with all pair-wise costs, DTW starts at the last entry and traces back the path of lowest accumulated cost until it reaches the first entry. This path yields the final point-by-point alignment and the final cost can be obtained from the matrix entry $D(N, M)$. Figure II.4 shows an example of two trajectories and the resulting alignment produced by the DTW algorithm. For simplicity, we assume that each state of a trajectory can be represented by a single value and use the absolute difference between two states as a distance measure for our example. The cost matrix shows the final matrix that contains the accumulated costs, while the red graph represents the final alignment.

4. Diffusion

4.1. Diffusion Process

The diffusion process (Ho et al., 2020; Sohl-Dickstein et al., 2015) defines two Markov chains: a forward and a reverse diffusion process. The forward process starts from real data, for example, an image, and gradually adds Gaussian noise over many steps until the original data is practically equivalent to an isotropic Gaussian.

The reverse process aims to invert the forward process and tries to reconstruct the data. The reverse process depends on the true data distribution $q(x_{t-1}|x_t)$, which would require knowing the entire data set without holes. Therefore, the model learns an approximation $p_\theta(x_{t-1}|x_t)$ using an NN. This NN is trained on samples from the forward process by trying to predict the added noise. This allows the model to later generate new samples by iteratively removing the predicted noise from pure Gaussian noise. As illustrated in Figure II.5, the forward process gradually transforms the original image on the right into pure Gaussian noise on the left, while the reverse process reconstructs the image by progressively removing the noise.

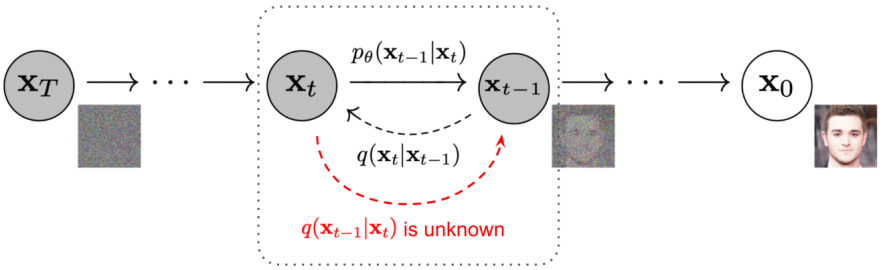


Figure II.5.: Markov chain of forward (reverse) diffusion process (image source: Weng (2021)).

4.2. Diffusion Policies

The diffusion process explained above is also usable in RL. It can be used to reconstruct the underlying action distribution from a standard Gaussian. Given a particular state and fully noised action, the learned neural network is able to generate an action according to the underlying distribution. Diffusion policies offer a great benefit because they can model the underlying distribution much more closely than Gaussian policies, which are the standard parameterization for continuous action spaces in RL. Thus, it is possible for them to model multiple modes instead of just one. This can be seen in the illustrative example shown in Figure II.6, where the red Gaussian distribution is not able to model the underlying Q-function as closely as the green distribution.

5. Diffusion-QL

Diffusion-QL (Wang et al., 2022) is an offline RL algorithm that tries to overcome a limitation of offline policy-regularization methods, which are restricted by weak policy classes. Typically, the policy is represented as a Gaussian distribution that

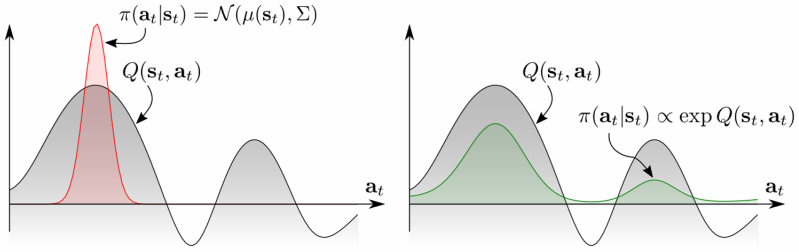


Figure II.6.: Unimodal vs. multimodal policy fit to $Q(s_t, a_t)$ (image source: Tang and Haarnoja (2017)).

cannot model multimodal distributions. This can result in the policy placing high density in the region between two modes. Regularizing a new policy towards such a policy is likely to negatively affect policy learning. To overcome this limitation, the authors perform policy regularization using diffusion models, specifically a multilayer perceptron (MLP) based denoising diffusion probabilistic model (DDPM (Ho et al., 2020)). The resulting denoising loss $\mathcal{L}_d$ serves as the behavioral cloning term, which on its own would not allow the learned policy to outperform the policy that generated the training data. To improve the policy beyond the behavior policy, the authors inject Q-value guidance into the reverse diffusion chain during training. This results in the following policy learning objective

$$\pi = \arg\min_{\pi_\theta} \mathcal{L}(\theta) = \mathcal{L}_d(\theta) + \mathcal{L}_q(\theta) = \mathcal{L}_d(\theta) - \alpha \cdot \mathbb{E}_{s\sim D, a^0\sim\pi_\theta}[Q_\phi(s, a^0)],$$

which consists of a linear combination of policy regularization and improvement. Here, a^0 is the action produced by the reverse diffusion chain. Since a^0 is reparameterized through that chain, the gradient of Q_ϕ is backpropagated through all diffusion steps. This is feasible here due to the small number of steps ($N = 5$). However, online methods such as DDiffPG and DIPO (Li et al., 2024; Yang et al., 2023) avoid this direct injection of Q-values because of the vanishing gradients that can occur when backpropagating through multiple diffusion steps.

6. DIPO

DIPO is introduced in a paper by Yang et al. (2023). The algorithm uses diffusion policies instead of unimodal policy representations to learn multimodal policies, with the goal of achieving better exploration and performance. One problem with diffusion policies is that the policy gradient theorem does not directly apply to them, as actions are generated through an iterative stochastic denoising process. Therefore, the policy gradient $\nabla_\theta J_\theta = \mathbb{E}_{s,a}[\nabla_\theta \log \pi_\theta(a|s) Q(a|s)]$ cannot be applied

directly. Computing the policy likelihood $\log \pi_\theta(a|s)$ is intractable because the actions are not drawn from any explicit parametric distribution. As a result, while the diffusion policy can fit the policy distribution, it cannot directly improve it. To overcome this limitation, the authors present the action gradient method $a_t \leftarrow a_t + \eta \nabla_a Q_\pi(s_t, a_t)$, which can be used to update and improve actions. They apply the action gradient to every state-action pair (s, a) stored in the replay buffer D, in order to improve the actions. Afterwards, the diffusion policy is fitted to the improved data in the replay buffer, resulting in an improved policy.

7. DDiffPG

The paper by Li et al. (2024) introduces DDiffPG, an algorithm that also uses diffusion policies to learn multimodal policies. In addition to DDiffPG, the authors introduce the diffusion policy gradient method, which, like DIPO, avoids backpropagating Q-values through the diffusion chain by computing a target action $a^{target} \leftarrow a + \eta \nabla_a Q(s, a)$ and fitting the policy to it. However, unlike DIPO, DDiffPG retains the original action a and stores the target action a^{target} alongside it rather than replacing it. This ensures that the transitions remain consistent with the MDP dynamics and the reward function when used for Q-learning. Building on this approach, the DDiffPG algorithm, depicted in Figure II.7, consists of four main stages, which are described below.

Stage 1 In the first stage, the agent interacts with the environment and collects trajectories, choosing actions and thereby exploring the state space. To encourage exploration, the method maintains an exploratory mode that is trained using intrinsic motivation. The intrinsic reward is based on the difference in novelty between the state before and after a transition and is given as $r^{int}(s, a, s') = \max(novelty(s') - \alpha \cdot novelty(s), 0)$. During interaction, the agent generates trajectories that contain states, actions, and the corresponding transition rewards. The resulting trajectories are then used for subsequent mode discovery and mode-specific learning.

Stage 2 In the second stage, the goal-reaching trajectories generated in the first stage are grouped into clusters, where each cluster will serve as one behavior mode. The applied clustering approach is hierarchical clustering in which each trajectory initially forms its own cluster. Clusters are then merged step by step based on a distance matrix computed using the DTW metric and a given threshold. Importantly, before computing the DTW distances, trajectories are represented as sequences of 2D coordinates. This implies a reliance on the expressiveness of the 2D coordinates in the underlying environment.

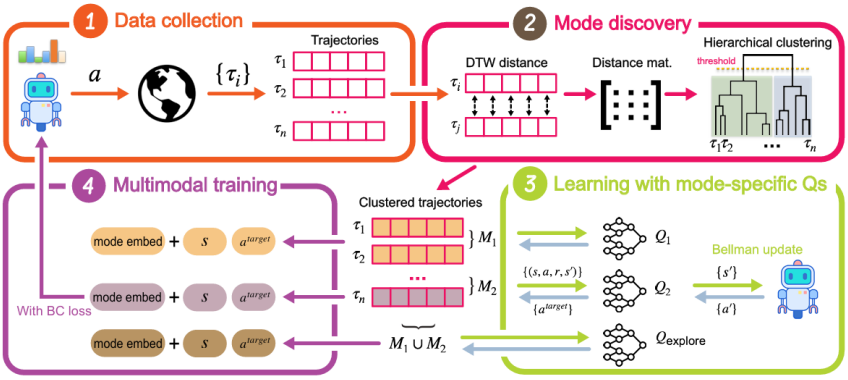


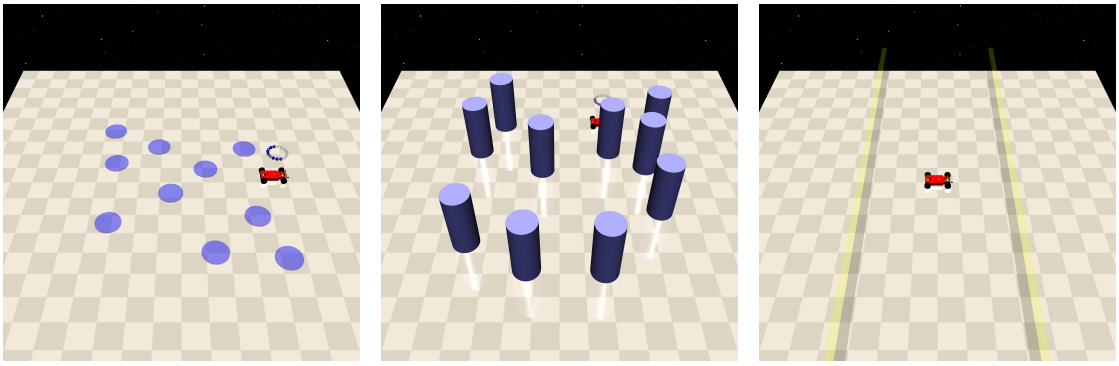
Figure II.7.: DDiffPG algorithm (image source: Learning Multimodal Behaviors from Scratch with Diffusion Policy Gradient (Li et al., 2024)).

Stage 3 Each mode created in the second stage gets its own mode-specific Q-function, which is now trained in the third stage of the algorithm. Every Q-function is updated using the transitions (s, a, r, s') from the mode/cluster to which it belongs. In addition to the mode-specific Q-functions, there is another one serving as an exploration mode. This exploration Q-function is trained on transitions collected from all modes together with the intrinsic reward. It ensures that exploration is decoupled from exploitation and the agent can go on exploring after already having found a goal. Having mode-specific Q-functions not only ensures ongoing exploration but also ensures the pursuit and preservation of rather suboptimal modes. It prevents the policy from collapsing into a single mode. In this stage, the algorithm creates multimodal data batches containing successful and unsuccessful trajectories from all different modes.

Stage 4 Stage four performs the policy update step using the multimodal data batches created in stage three. Those data batches contain states together with the corresponding target action, computed via the diffusion policy gradient. The policy is updated with the behavioral cloning (BC) (Li et al., 2024) objective. During the policy update, Li et al. (2024) attach a latent variable to the state, which serves as a unique identifier. By setting or masking the latent variable, one can either explicitly control a selected mode or let the agent choose freely.

8. Safety Gymnasium

Safety Gymnasium is a safe RL environment presented in the paper by Ji et al. (2023). In this environment, agents try to solve different navigation tasks while still satisfying safety constraints. The environment offers a wide variety of navigation



(a) Hazards (b) Pillars (c) Sigwalls

Figure II.8.: Safety Gymnasium environments.

tasks, presenting different obstacles to learning. For example, Figure II.8a shows circular purple zones that the agent should avoid traversing. Crossing these hazards incurs a fixed cost. Figure II.8b shows obstacles that are solid and cannot be passed by the agent. These obstacles can completely block certain paths, meaning that the agent has to navigate around them. In Figure II.8c, we can see two transparent walls limiting the agent's space. These sigwalls can be generated to create a smaller subspace within the environment in which the agent can move safely. If it crosses the border and moves outside of these restrictions, it is penalized with a cost.

Chapter III.

Methodology

1. Reproduction of Original DDiffPG Experiments

We used the original implementation by the authors available online[1] to reproduce the experiments. Unfortunately, we were not able to reproduce the exact experiment results mentioned in the paper using the specified hyperparameters. To clarify these discrepancies, we contacted the authors [2]. The detailed results can be seen in Section IV.3.1.

2. Adapting DDiffPG to Safety Gymnasium

A major drawback of the existing DDiffPG algorithm is that it directly works on the agent's (x, y)-coordinates to calculate the distances between the trajectories for clustering. In the AntMaze environment, these coordinates can be extracted from the state representation, making them suitable for differentiating between trajectories. However, this is not necessarily the case in other environments. Furthermore, it is not trivial to determine which features are best suited to distinguish between modes in different environments. To demonstrate this limitation and investigate the applicability of DDiffPG to a different environment, we also adapted the code to work with Safety Gymnasium. For this purpose, we manually extracted the coordinates from the environment after each step. This approach

[1] `https://github.com/supersglzc/ddiffpg`

[2] This is not expected to significantly affect the proposed work. In our experiments, the state coverage plots consistently reported lower values than those presented in the original paper; however, the corresponding state coverage maps appeared visually identical to those reported by the authors, suggesting that the discrepancy may be caused by differences in the plotting functions rather than the underlying experiments. We also encountered a minor rounding error that eventually caused training to crash. The authors provided us with the original dependency versions, noting that the rounding error could be caused by a version mismatch in PyTorch.

cannot be applied in general, as different environments represent and handle coordinates differently. Therefore, DDiffPG is not directly applicable to arbitrary environments.

A second limitation resulting from the reliance of DDiffPG on (x, y)-coordinates as the complete state representation for distance calculation is the inability to distinguish between safe and unsafe states. In the AntMaze environment, this does not pose a problem because the agent cannot experience unsafe behavior. In Safety Gymnasium, on the other hand, the agent receives a cost after each step that indicates how safe the chosen action is. This information is completely lost when applying the adapted version of DDiffPG state clustering to Safety Gymnasium.

Additionally, the threshold chosen for clustering in DDiffPG is tuned specifically for each AntMaze environment. Therefore, keeping the default threshold also poses a problem when applying the algorithm to a different environment. We also tried different adaptive thresholds, such as percentile-based thresholds, to make the threshold independent of the environment. However, these were also not applicable to Safety Gymnasium, as the resulting thresholds are not suitable for the environment. Therefore, they have to be replaced by the same threshold that is also used for the proposed Latent DDiffPG method described in the following section.

3. Proposed Latent DDiffPG Method

Our proposed extension to the DDiffPG algorithm introduces a flexible latent embedding module that can incorporate different representation learning methods, such as ICM, EMI or CTEC. Concretely, we introduce two encoder networks: a frozen encoder ϕ_{frozen} and a trainable encoder ϕ_{train}, both mapping environment states s to latent embeddings. The trainable network is updated jointly with the Q-functions during the agent's gradient-based updates performed on each training iteration. In contrast, the frozen encoder is not updated regularly by gradient descent. To keep the latent representations stable while still allowing for improvements over time, the weights of the trainable network are copied to the frozen encoder at fixed intervals controlled by a hyperparameter.

The second major code modification directly impacts the clustering stage. In the original implementation, clustering is performed on 2D coordinates extracted from the state. In our implementation, this step is replaced by a latent representation lookup. Specifically, instead of extracting (x, y) from the environment state, s is passed through the frozen encoder to obtain a latent vector $\phi_{frozen}(s)$, which is then used as input to the distance function for clustering. We also modify the clustering function by changing the criterion option from *distance* to *maxclust*. Previously, the clustering function received a fixed threshold value and cut the dendrogram

at this particular point. If no threshold is specified, the original implementation automatically calculates one as 0.7 times the maximum linkage distance. With the *maxclust* criterion, the threshold is instead determined automatically such that the resulting clustering contains no more clusters than the specified value. This reduces the risk of the number of clusters becoming excessively large due to a poorly chosen threshold.

Apart from these modifications, the remaining components of DDiffPG are left unchanged.

4. Offline and Online Embedding Training

Before applying the proposed modifications in the online RL setting, we first trained and evaluated the embeddings offline. For this purpose, we used a pretrained policy from the original DDiffPG algorithm and executed all its individual modes, generating 1000 trajectories per mode. Based on these generated trajectories, we trained different embedding networks with different hyperparameter configurations. The resulting latent representations were then used to perform clustering to evaluate the learned embeddings. This allowed us to identify suitable network configurations that could be used for the online policy training.

In a first stage of online training, the selected pretrained embedding network was used together with the RL policy. Finally, the embedding network was no longer fixed, but was trained jointly with the RL policy, as described in Section III.3. This allows the embedding to adapt to the representations learned during the policy training.

5. Cost Integration

Initially, the cost generated by the environment was added as a negative reward, completely excluding the signal from the clustering process. As a result, the embedding did not have information on whether a state transition was safe or unsafe. To make the embedding aware of the cost, we introduced an additional cost-related signal. For ICM and EMI, this was done using an auxiliary network, while for CTEC the cost information was incorporated directly into the contrastive loss.

In ICM and EMI, we introduced a small NN with an input size of twice the embedding size and a hidden layer of size 32, which outputs a one-dimensional value. The cost network receives the embeddings $\phi(s_t)$ and $\phi(s_{t+1})$ of the current state and the next state. Based on these embeddings, the network predicts the cost associated with the transition. The prediction error (*cost loss*) of this cost

network is calculated using the cross-entropy loss, as the cost is a binary value of either zero or one. The resulting training loss is a weighted combination of the representation-learning loss and the *cost loss*, given for ICM by

$$\beta \cdot fw_loss + (1 - \beta) \cdot inv_loss + w_{cost} \cdot cost_loss$$

and for EMI by

$$pred_loss + \lambda_{err} \cdot error_loss + \lambda_{info} \cdot \mathcal{L}_{info} + w_{cost} \cdot cost_loss.$$

As the new loss is backpropagated, the embedding network also receives gradients from the cost prediction. Therefore, it is encouraged to retain the information needed to correctly predict the costs. This should allow the representation to distinguish between transitions with different costs.

For CTEC, the cost information was incorporated directly into the contrastive learning objective. The InfoNCE loss is extended with a cost-dependent weighting term, which gives samples with lower cost a greater influence on the contrastive learning process. The original CTEC loss is given by

$$L_{CTEC} = -\frac{1}{N} \sum_{i=1}^{N} \log(p_i).$$

To incorporate the cost signal, this loss is extended to

$$L_{CTEC}^{cost} = -\frac{1}{N} \sum_{i=1}^{N} w_{cost,i} \cdot \log(p_i),$$

where the cost weight for sample i is defined as

$$w_{cost,i} = e^{-\alpha \cdot c_i},$$

with c_i denoting the cost of sample i and α a hyperparameter controlling the influence of the cost. Since the cost is binary, a transition with $c_i = 0$ receives a weight of one, while a transition with $c_i = 1$ receives a lower weight of $e^{-\alpha}$. Thus, lower-cost transitions contribute more strongly to the contrastive learning objective. During backpropagation, this weighting directly scales the gradient contributed by each transition. No separate cost-prediction network is introduced. In this way, the cost signal directly influences learning and encourages the network to prioritize the information from lower-cost transitions and, therefore, safer experiences.

Chapter IV.

Empirical Evaluation

In order to compare the original DDiffPG and our Latent DDiffPG algorithm, we ran experiments on two sets of environments: the AntMaze environment, which was used to evaluate the algorithm in the original paper, and the Safety Gymnasium environments, which we introduced to include the safety aspect and test transferability. The experiments were conducted both offline and online to find the most suitable embedding network that can solve the task most reliably. We compare the algorithms with respect to their return, stability, and mode separation ability.

1. Environment Setup

AntMaze The *AntMaze* environment (Figures IV.1a and IV.1b) serves as the main comparison benchmark in the original paper by Li et al. (2024). It provides a high-dimensional and continuous control task. The goal is to control the leg joints of the ant to reach the goal position. It offers multiple possible solutions with multiple goals or different ways of solving the task (Figures IV.2a and IV.2b).

Safety Gymnasium Pillars The *Safety Gymnasium Pillars* environment (Figure IV.1c) is a custom Safety Gymnasium environment. It contains three pillar obstacles, which are cylindrical, solid obstacles that the agent has to navigate around, placed in a line between the start and the goal. The agent can choose between four different modes to reach the goal: Mode 1 passes below the obstacles, Modes 2 and 3 pass through the two passages between them, and Mode 4 passes above the obstacles (Figure IV.2c). These distinct routes make the environment suitable for evaluating the algorithm's ability to find and separate different modes.

Safety Gymnasium Hazards Following the same layout as Safety Gymnasium Pillars, this environment (Figure IV.1d) uses identical placement of the start,

goal, and obstacles. Instead of pillars, it uses hazards, which are cylindrical, non-solid obstacles. In contrast to pillars, the agent can pass through those obstacles, resulting in a cost for the corresponding transition. This environment is used to evaluate the algorithm's ability to estimate the costs associated with different modes.

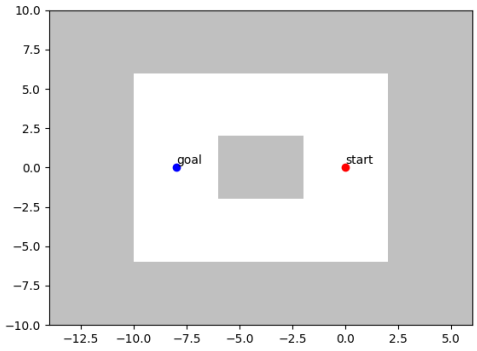
Safety Gymnasium Wall In this environment (Figure IV.1e), a different obstacle layout is used. It consists of pillars that intersect in a way that makes passing between them impossible, creating a larger wall between the start and the goal. This provides the agent with only two valid modes for reaching the goal, resulting in more spatially separable transitions. Mode 1 passes below the wall and Mode 2 passes above (Figure IV.2d).

Safety Gymnasium Random Pillars In the *Safety Gymnasium Random Pillars* (Figure IV.1f) environment, the distance between the start and the goal increases. Five randomly placed pillars are positioned between the start and the goal, offering the agent multiple paths to reach the goal. This environment tests the agent's ability to explore different solutions and also keep them separate.

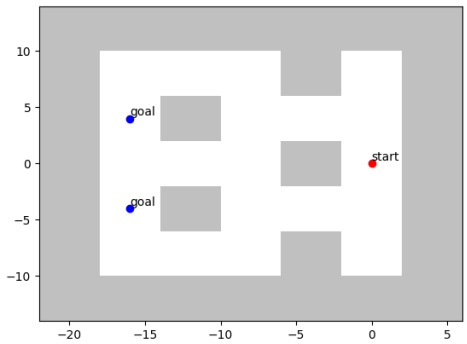
2. Hyperparameters and Training Setup

Offline Training We generated a set of trajectories by executing a policy trained with the original DDiffPG, covering a wide variety of states. ICM, EMI, and CTEC were then trained offline on the same set of trajectories for 25 epochs. The embeddings were initially trained using the standard hyperparameters reported in the respective papers. We additionally experimented with selected hyperparameters where the default settings did not provide suitable results. These parameters were selected manually over a small range. In the offline stage, we only used the Safety Gymnasium Pillars and AntMaze environments to provide consistent evaluation environments to compare the different approaches. The main purpose of the offline experiments was to investigate which features the different embeddings focus on and how this applies to our use case. We also compared the performance of the different embeddings in the named environment. Based on these results, suitable combinations of networks and parameters were chosen for the following experiments.

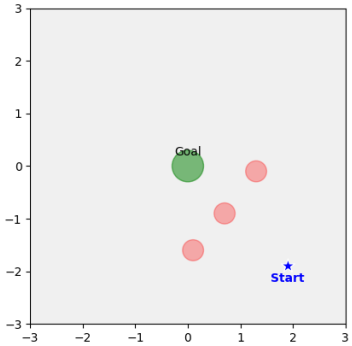
Online Training In this stage of the experiments, we used the findings from the offline training stage to evaluate the most promising configurations in the different environments. Each selected network was trained in one of the environments described in Section IV.1 using ten different seeds. This stage was used to evaluate the performance of the selected configurations in the online training and to compare



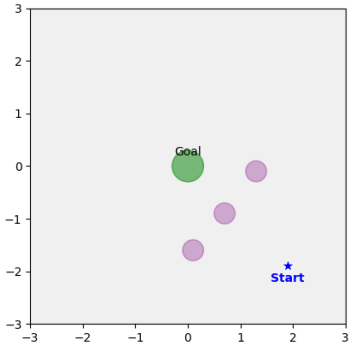
(a) AntMaze-v1



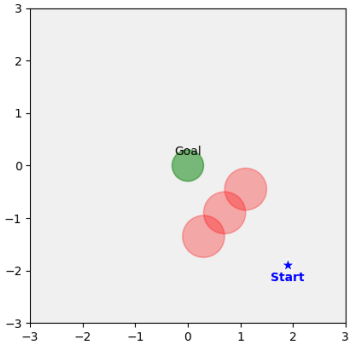
(b) AntMaze-v4



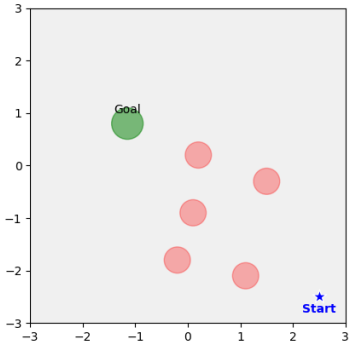
(c) Pillars



(d) Hazards

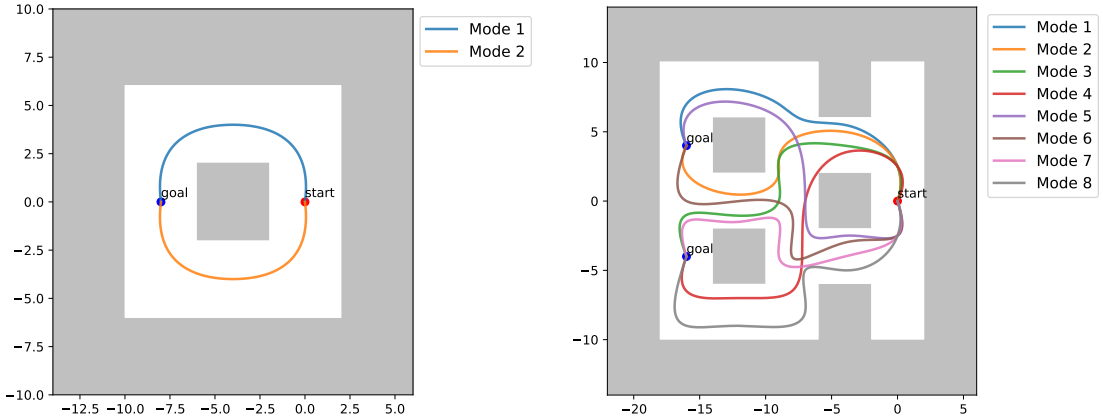


(e) Wall



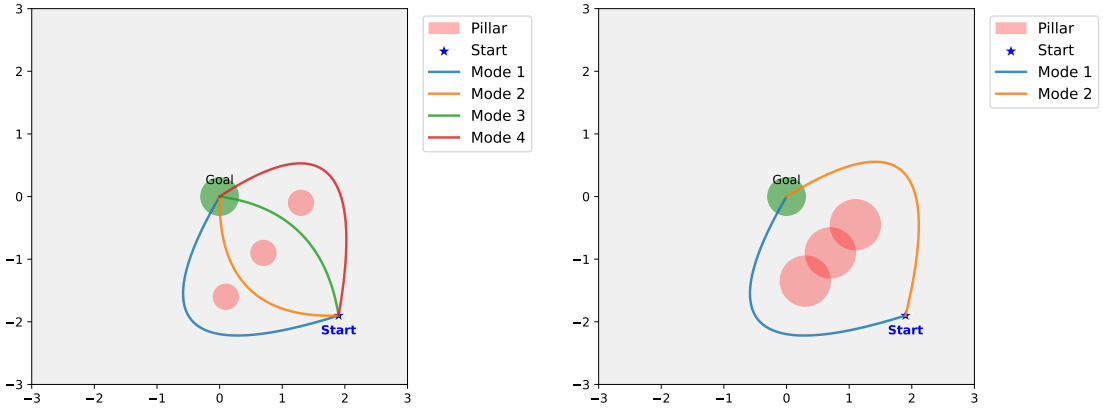
(f) Random Pillars

Figure IV.1.: Environments used in the experiments.



(a) AntMaze-v1

(b) AntMaze-v4



(c) Pillars

(d) Wall

Figure IV.2.: Predefined behavior modes.

their ability to solve the task with that of the original algorithm. The multiple seeds allow for a more reliable assessment of the results.

3. Standard DDiffPG Results

In the following experiments, a behavior mode is considered successfully separated if trajectories moving substantially differently through space are assigned to distinct clusters. Here, substantially different refers to trajectories following different paths to the goal or exploring different regions of the environment, whereas trajectories following similar paths or exploring similar regions should be assigned to the same cluster.

3.1. Reproduction of DDiffPG Results

Li et al. (2024) evaluate their approach on four AntMaze tasks and four robotic control tasks, comparing to DIPO (Yang et al., 2023), Diffusion-QL (Wang et al., 2022), SAC (Haarnoja et al., 2018) and RPG (Huang et al., 2023). All of those tasks are high-dimensional control tasks. The experiments show that DDiffPG achieves performance comparable to the baselines in terms of both return and state coverage, while having the advantage of discovering and mastering multiple modes. SAC, RPG, Diffusion-QL and DIPO (the latter two are diffusion-based approaches) collapse to one mode at a certain point. However, DDiffPG is capable of detecting multiple paths through the maze and even keeping the suboptimal ones. The method achieves a comparable return to the other algorithms due to its mode balancing ability.

When recreating the original results from Li et al. (2024), we found that the state coverage from our experimental runs, which can be seen in Figure IV.3, is consistently lower than the reported state coverage across all four AntMaze environments. Interestingly, the result is the same when running the baseline DIPO. Since these results originate from the authors' unchanged code, we contacted them for clarification. The details can be seen in Section III.1. In terms of mode discovery and separation, the results are consistent with the figures in the paper.

3.2. Standard DDiffPG on Safety Gymnasium

In the Safety Gymnasium Pillars environment, the original algorithm fails when no clustering threshold is specified, and a threshold is calculated automatically based on the linkage distances. The number of resulting clusters either grows rapidly, reaching approximately 500 modes, or the clustering collapses completely, merging all clusters into a single cluster. The original code fixes a threshold for

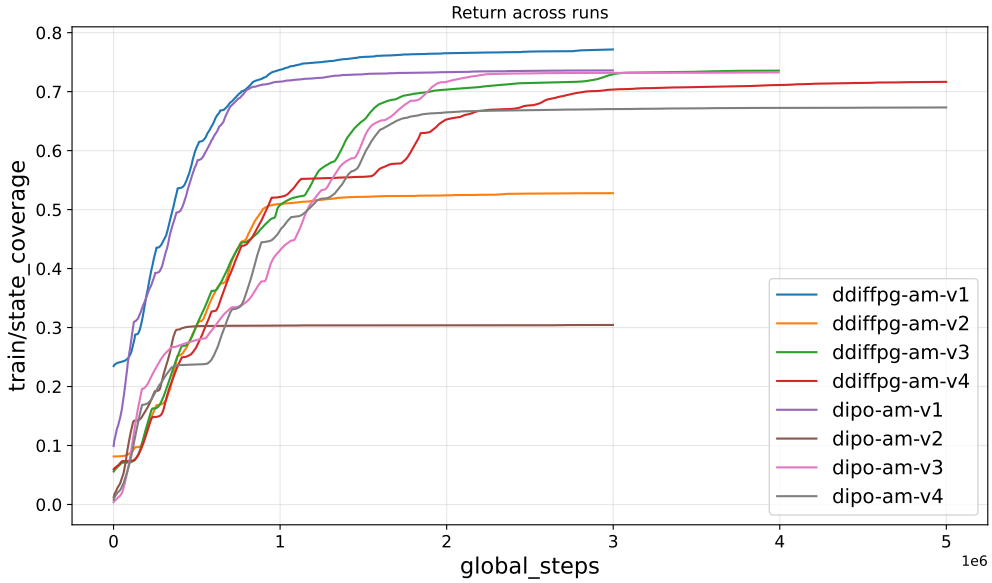


Figure IV.3.: State coverage of DDiffPG and DIPO on AntMaze, from our reproduced experiments.

the different AntMaze tasks, but these thresholds are not applicable in Safety Gymnasium due to the different scales of the state values. We experimented with different thresholds and found that a value of 15.0 was able to keep the number of clusters in a reasonable range. However, this tuned threshold is again fixed for the specific custom environment. Therefore, for the following experiments, we decided to apply a different thresholding method (*maxclust*), as already described in Section III.3.

The experiments show that the original DDiffPG is able to find the goal for all ten seeds in the Safety Gymnasium Pillars environment. This is reflected in the positive rewards achieved by all seeds towards the end of training, as shown in Figure IV.4. However, it is not able to consistently find and clearly separate expected behavior modes. In all runs, the agent commits to one mode toward the goal and does not retain additional goal-reaching modes, as can be seen in Figure IV.5c. Thus, in contrast to the results reported in the original paper for AntMaze, DDiffPG collapses to a single goal-reaching mode before the end of training.

In Section IV.1, we define four distinct modes in the Safety Gymnasium Pillars environment, which can be seen in Figure IV.2c. Table IV.4 shows how many seeds discover each of these predefined modes and whether the corresponding trajectories reach the goal.

At the beginning of the training runs, we observe that, for some seeds, the algorithm is able to separate trajectories belonging to different behavior modes,

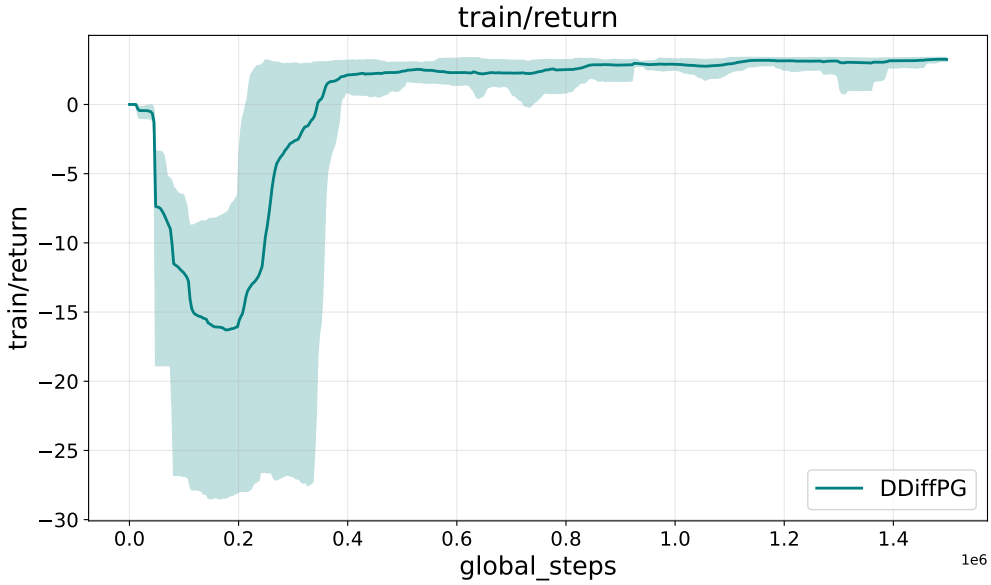


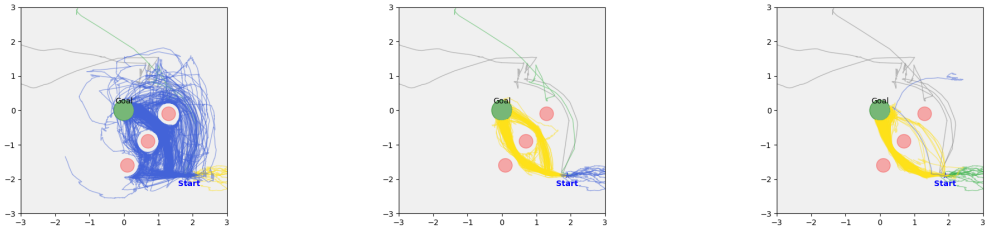
Figure IV.4.: Return of DDiffPG on the Safety Gymnasium Pillars environment.

while on others it fails to provide any meaningful clustering. Three snapshots from one run are shown in Figure IV.5. After 200,000 steps (Figure IV.5a), the algorithm explores multiple paths but fails to keep them separated. Later in the same run, at 700,000 steps (Figure IV.5b), the algorithm discards all the goal-reaching trajectories that do not pass through the center and only separates trajectories going in substantially different directions. The final snapshot at 1.5 million steps (Figure IV.5c) shows that the agent remains committed to this single mode. The algorithm appears to be able to separate trajectories that differ substantially in length or direction, but struggles to distinguish trajectories that are spatially similar while following different paths around an obstacle. In Figure IV.6, we can see another example of the algorithm being able to separate the trajectories in different locations but not keeping separate goal-reaching modes.

4. Latent DDiffPG Results

4.1. Offline Latent DDiffPG

During offline training, we observe that CTEC has the most success encoding coordinate information, whereas ICM and EMI have difficulty preserving this information. This may stem from the fact that ICM and EMI train their encoder using only one transition at a time, focusing on the result of an action taken between a particular state s_t and its direct successor s_{t+1}. While ICM aims to preserve the



(a) 200k steps (b) 700k steps (c) 1.5 million steps

Figure IV.5.: Behavior modes discovered by DDiffPG on Safety Gymnasium Pillars (run 5).

information influencing the agent's action choice, EMI tries to maximize the mutual information between the two states. In both cases, the coordinate information appears to be discarded, as it seems to be irrelevant for making immediate action decisions. CTEC, on the other hand, focuses on preserving temporal structure by sampling a future state that does not necessarily have to be a direct successor of s_t. As a result, CTEC observes meaningful changes in the coordinates over the course of a trajectory, encouraging it to preserve this information.

To evaluate how well the coordinate information is encoded, we train a linear probe and an MLP probe to predict the (x, y)-coordinates from a given state embedding. As shown in Table IV.1, CTEC encodes the coordinates in the most accessible manner compared to ICM and EMI, as indicated by its lower mean squared error (MSE) and very high R^2 scores, particularly for the MLP probe.

Table IV.1.: Coordinate prediction performance of the three state embeddings using linear and MLP probes.

	Linear Probe			MLP Probe		
Embedding	MSE	R_x^2	R_y^2	MSE	R_x^2	R_y^2
ICM	33.9	0.006	0.007	21.7	0.43	0.25
EMI	31.2	0.12	0.03	20.4	0.46	0.29
CTEC	4.7	0.87	0.84	0.03	0.99	0.99

Another interesting observation is made by comparing different trajectories embedded in the respective latent space using principal component analysis (PCA). For ICM, trajectories from AntMaze (Figure IV.7a) and Safety Gymnasium (Figure IV.8a) do not exhibit any clear structure. They mostly collapse or spread randomly through the latent space, making it difficult to infer meaningful patterns. EMI trajectories from AntMaze, on the other hand, exhibit a circular structure

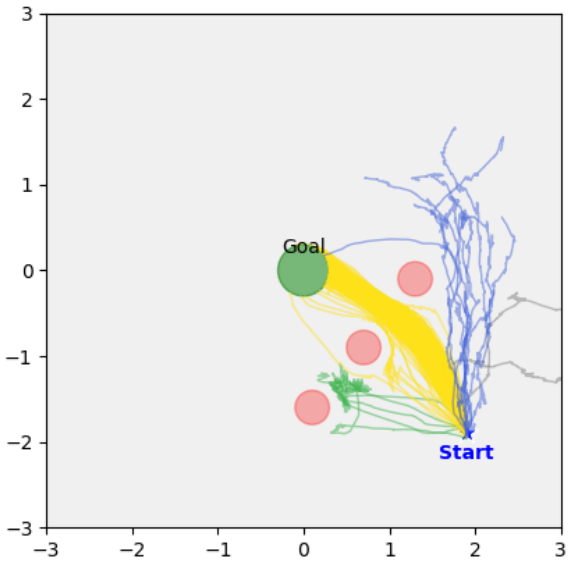
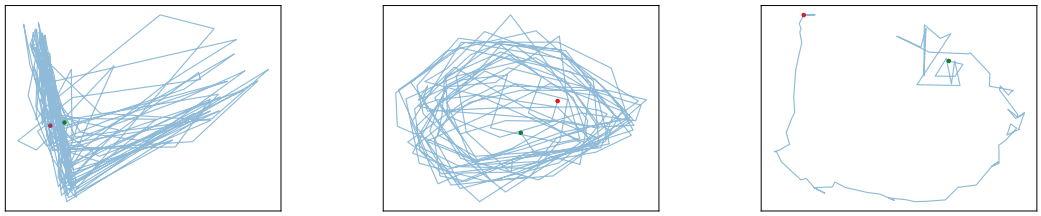


Figure IV.6.: Behavior modes discovered by DDiffPG on Safety Gymnasium Pillars (run 6).

(Figure IV.7b), while the trajectories from Safety Gymnasium (Figure IV.8b) again show no clear structure. The circular structure observed in AntMaze could be explained by the fact that the original state mainly contains joint angles, which may lead the network to capture the ant's circular movements throughout a trajectory. In Safety Gymnasium, however, the different state representation appears to result in a collapse of the trajectory once again.

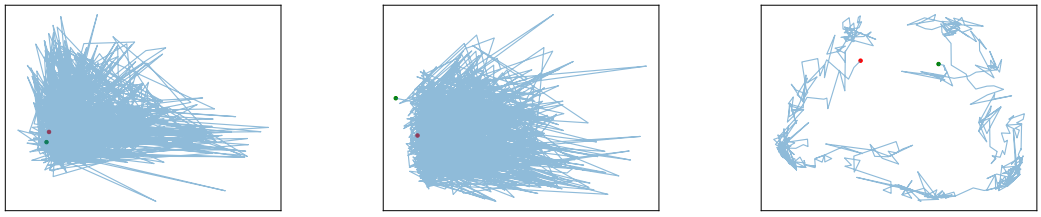
CTEC is the only embedding method in which the trajectories exhibit a structure that reflects their actual progression through the state space. In both AntMaze (Figure IV.7c) and Safety Gymnasium (Figure IV.8c), the embeddings form distinct paths through the latent space that correspond to the progression of the trajectories. This seems to be influenced by the fact that CTEC is able to preserve the progression of trajectories, in contrast to ICM and EMI, which focus only on individual transitions.

Evaluating the embeddings trained offline showed that CTEC is able to identify distinct clusters corresponding to different behavior modes, whereas ICM and EMI mainly collapse all trajectories into a single large cluster. As shown in Figure IV.9, CTEC distinguishes multiple modes, while EMI collapses them into one large mode. CTEC groups all goal-reaching trajectories, while also separating trajectories that



(a) ICM (b) EMI (c) CTEC

Figure IV.7.: Trajectories of Latent DDiffPG in the AntMaze-v4 environment, projected into 2D using PCA.



(a) ICM (b) EMI (c) CTEC

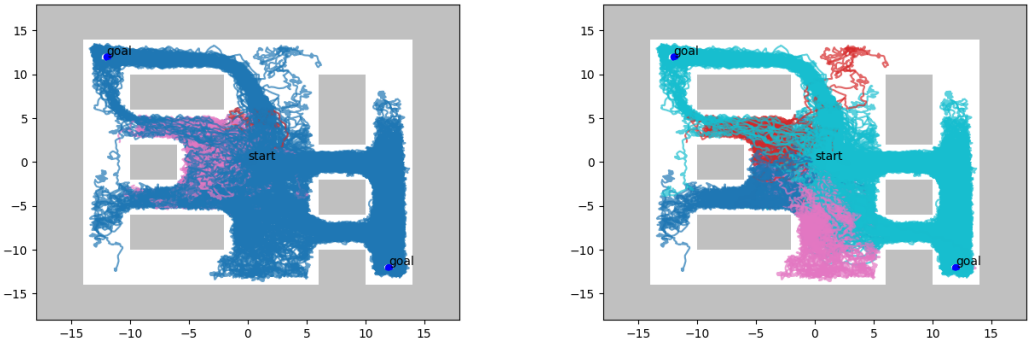
Figure IV.8.: Trajectories of Latent DDiffPG in the Safety Gymnasium Pillars environment, projected into 2D using PCA.

move through different regions of the state space. The results for ICM are similar to those for EMI.

These findings from the offline setting motivate us to continue the online experiments primarily using the CTEC embedding.

4.2. Online Latent DDiffPG on AntMaze

We evaluate our latent extension using the CTEC network in two of the benchmark environments that the paper by Li et al. (2024) uses: AntMaze-v1 and AntMaze-v4. In AntMaze-v1, every run is able to find the goal and receive a similar return to the original algorithm (Figure IV.10a). The latent algorithm is able to separate the two modes in 8 out of 10 seeds. Table IV.2 shows how many of the 10 seeds discover a predefined mode. Mode 1 corresponds to the path above the wall, while Mode 2 corresponds to the path below it (Figure IV.2a). However, this separation occurred relatively early in training, as shown in Figure IV.11a, and the algorithm is not able to preserve the expected clusters throughout the remainder of the training.



(a) EMI (b) CTEC

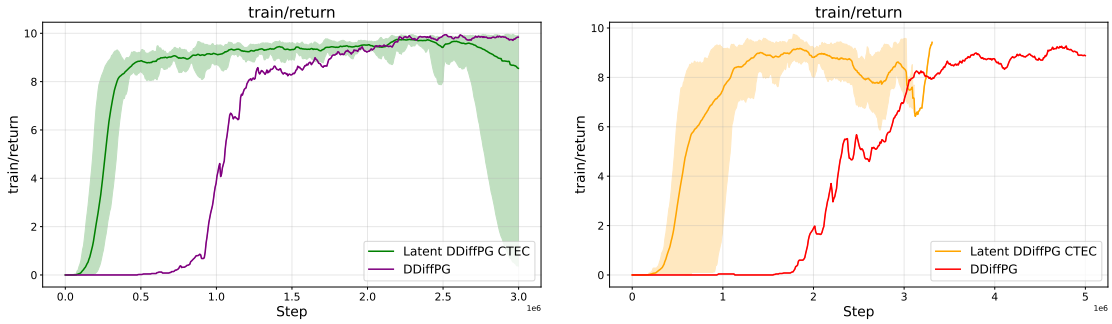
Figure IV.9.: Resulting clusters of trajectories on AntMaze-v4, obtained by applying hierarchical clustering to EMI and CTEC representations, following the approach used in Latent DDiffPG.

After some time, the separated clusters consistently collapse into one, which can be seen in Figure IV.11b. One possible reason for this is that, after the embedding is updated, the algorithm recalculates all the distances. These distances can change substantially with the new embedding, potentially leading to a collapse if the embedding loses important information. A second possible explanation is that the trajectory structure changes over time, as the agent discovers differently shaped paths that can no longer be separated by the current threshold.

Table IV.2.: Number of seeds (out of 10) discovering each mode on AntMaze-v1 when running Latent DDiffPG CTEC, shown both at the final training step and at any point during training.

Mode	Final Training Step	During Training
Mode 1 only	3	1
Mode 2 only	1	0
Mode 1 and 2, separated	0	8
Mode 1 and 2, merged	6	1

In AntMaze-v4, we observe similar results. The latent algorithm achieves a comparable return to the original algorithm in this environment (Figure IV.10b). At the beginning, the algorithm is able to separate the different modes it explores (Figure IV.12a). However, as training progresses, it consistently merges the separated modes or completely discards modes that had previously been discovered



(a) AntMaze-v1 (b) AntMaze-v4

Figure IV.10.: Return over training steps for original DDiffPG and Latent DDiffPG CTEC on AntMaze-v1 and AntMaze-v4

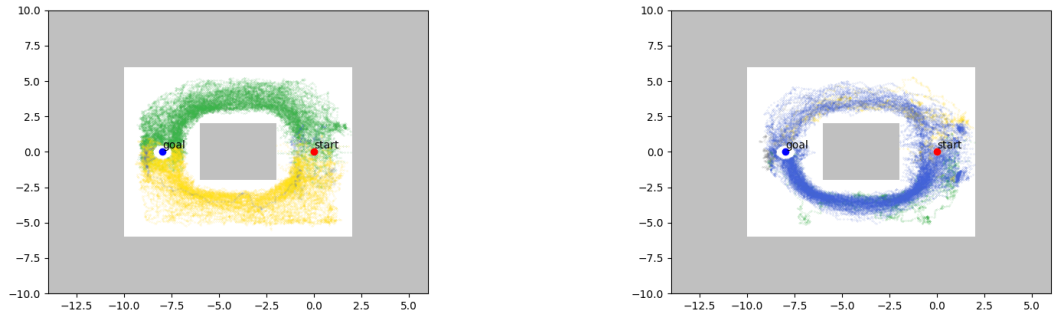
(Figure IV.12b). For AntMaze-v4, we also define eight distinct behavior modes. Modes 1-4 pass above the first wall, while Modes 5-8 pass below it. Modes 1 and 2 reach the upper goal, with one passing above and the other below the second obstacle, while Modes 3 and 4 follow the same structure to reach the lower goal. Modes 5-8 are defined following the same pattern (Figure IV.2b). Table IV.3 shows how many of the 10 seeds discover each mode. Modes 4 and 5 are never discovered, as they present behaviors that are also not retained by the original algorithm.

Table IV.3.: Number of seeds (out of 10) discovering each mode on AntMaze-v4 when running Latent DDiffPG CTEC, shown both at the final training step and at any point during training.

Mode	Final training step	During Training
1	2	3
2	3	5
3	1	3
6	3	6
7	0	2
8	5	6

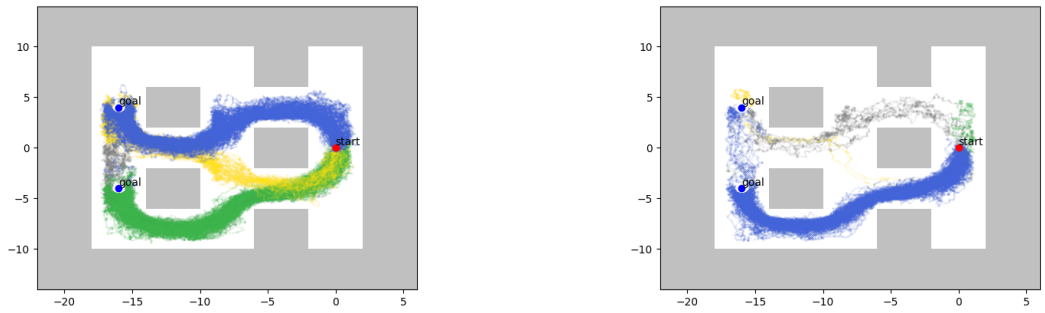
4.3. Online Latent DDiffPG on Safety Gymnasium

Safety Gymnasium Pillars Latent DDiffPG in combination with ICM's embedding network has difficulty finding useful cluster assignments across all ten seeds. It mostly commits to a single mode towards the goal and clusters all trajectories



(a) 250k steps (b) 2.6 million steps

Figure IV.11.: Behavior modes discovered by Latent DDiffPG CTEC on AntMaze-v1 (run 10).



(a) 1.2 million steps (b) 2.2 million steps

Figure IV.12.: Behavior modes discovered by Latent DDiffPG CTEC on AntMaze-v4 (run 10).

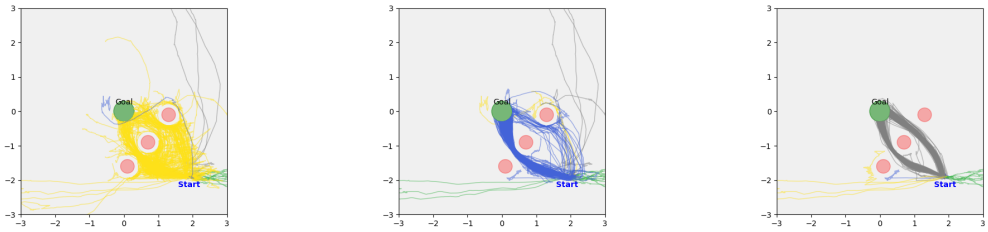
together. The additional clusters do not provide any meaningful separation, as can be seen in Figure IV.13. This behavior remains mostly unchanged throughout training without any clear improvement over the later steps. ICM does not encode coordinate information in an accessible manner, as we have already established from the results in Table IV.1. This missing information makes it difficult for the clustering algorithm to distinguish trajectories moving through different parts of the state space. Instead, the network mainly encodes information relevant to the agent's action decisions, which does not provide a suitable basis for clustering trajectories according to their location in the state space.

Using EMI as the embedding network already improves the results, leading to more meaningful cluster assignments. Although EMI is unable to consistently



(a) 500k steps (b) 1.5 million steps

Figure IV.13.: Behavior modes discovered by Latent DDiffPG ICM on Safety Gymnasium Pillars (run 4).

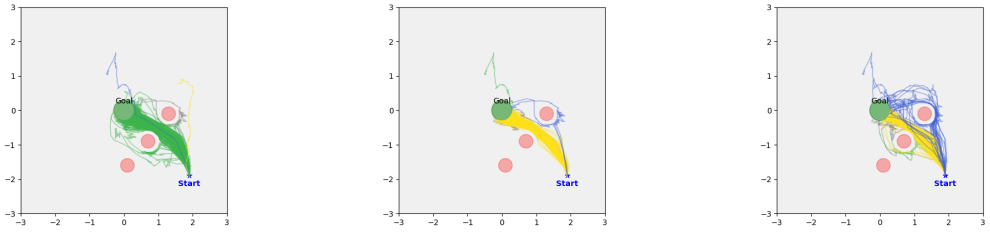


(a) 250k steps (b) 500k steps (c) 1.5 million steps

Figure IV.14.: Behavior modes discovered by Latent DDiffPG EMI on Safety Gymnasium Pillars (run 9).

identify goal-reaching modes, it is able to separate the trajectories more clearly than ICM. As shown in Figure IV.14, EMI is able to separate trajectories with larger differences, distinguishing between goal-reaching trajectories and trajectories moving to the right or the left from the start. Thus, compared to ICM, EMI provides a latent representation in which DDiffPG is able to find a greater diversity of behavior modes.

In online experiments, CTEC generally produces results similar to EMI and is able to distinguish between trajectories with larger behavioral differences. Regarding our predefined modes, Table IV.4 shows how many of the 10 seeds commit to each goal-reaching mode. CTEC primarily commits to a single mode and only differentiates between groups of trajectories with substantial differences, rather than preserving distinct but behaviorally similar goal-reaching modes. This behavior is consistent with the mode collapse observed in the original DDiffPG experiments in Safety Gymnasium. However, for one seed, it comes closest to identifying and



(a) 250k steps (b) 600k steps (c) 1.5 million steps

Figure IV.15.: Behavior modes discovered by Latent DDiffPG CTEC on Safety Gymnasium Pillars (run 6).

maintaining two separate goal-reaching modes. In Figure IV.15b, it identifies several trajectories at the top right of the obstacle as a separate mode and continues to assign additional trajectories to this mode. At the end of training (Figure IV.15c), the clustering is close to establishing two distinct behavior modes, which neither ICM, EMI, nor the original DDiffPG is able to achieve over the course of ten seeds.

Table IV.4.: Number of seeds (out of 10) discovering each mode for Latent DDiffPG CTEC and original DDiffPG on Safety Gymnasium Pillars. A mode is categorized as goal-reaching if the majority of its trajectories terminate in the goal state, and non-goal-reaching otherwise.

	Latent DDiffPG CTEC		Original DDiffPG	
Mode	goal-reaching	non-goal-reaching	goal-reaching	non-goal-reaching
1	0	3	0	4
2	3	1	5	0
3	8	0	4	0
4	1	1	2	2

Safety Gymnasium Wall In the Safety Gymnasium Wall environment, Latent DDiffPG with CTEC is also able to separate clusters that can be identified as different behavior modes (Figure IV.2d) in most of the ten seeds. In this environment, the algorithm has the possibility of finding two behavior modes reaching the goal that are spatially more distant than the modes in Safety Gymnasium Pillars. Table IV.5 shows how many seeds maintain each mode. It shows that three out of the ten seeds are unable to separate the two modes, while an equal number successfully maintains both distinct goal-reaching modes until the end of



(a) 650k steps (run 4)

(b) 1.5 million steps (run 10)

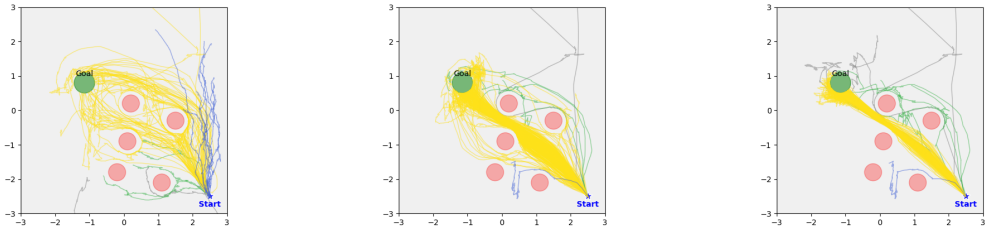
Figure IV.16.: Behavior modes discovered by Latent DDiffPG CTEC on Safety Gymnasium Wall.

training. The remaining four seeds commit to one of the two modes. Figure IV.16b shows that the algorithm is able to separate the two goal-reaching modes, going around either the left or right side of the obstacle. Figure IV.16a shows how the algorithm again identifies trajectories going to the left (blue cluster), moving to the right towards the goal (yellow cluster), circling around the goal (gray cluster), and remaining in front of the obstacle (green cluster).

Table IV.5.: Number of seeds (out of 10) discovering each mode for Latent DDiffPG CTEC on Safety Gymnasium Wall. A mode is categorized as goal-reaching if the majority of its trajectories terminate in the goal state.

Mode	goal-reaching
Mode 1 only	1
Mode 2 only	3
Mode 1 and 2, separated	3
Mode 1 and 2, merged	3

Safety Gymnasium Random Pillars In this environment, the algorithm has difficulty identifying and preserving multiple goal-reaching modes. Mostly it sticks to a single mode through the obstacles and assigns the remaining trajectories to different clusters. Figure IV.17 shows the progression of the training for one seed. We can see that the algorithm is able to roughly separate the trajectories based on their direction and position in the space. Particularly during the early exploration stages of the training, the algorithm is unable to distinguish the



(a) 100k steps (b) 500k steps (c) 1.5 million steps

Figure IV.17.: Behavior modes discovered by Latent DDiffPG CTEC on Safety Gymnasium Random Pillars (run 1).

different paths around the obstacles, leading it to discard many of them. This makes it more difficult to establish and maintain multiple goal-reaching modes. One possible explanation for this mode collapse in the Safety Gymnasium Random Pillars environment is the chosen threshold and the distance calculation. In this environment, trajectories that pass an obstacle on different sides can still share a large portion of their paths. As a result, the distance between them may remain relatively small because they intersect for a considerable amount of time. Our chosen threshold may therefore not be sensitive enough to distinguish between such trajectories.

5. Cost-aware Latent DDiffPG Results

The experiments applying Latent DDiffPG in the Safety Gymnasium Hazards environment show that the cost integration described in Section III.5 does not improve the separation of trajectories based on their safety. The algorithm is again able to separate modes based on trajectories' locations in space, but it is unable to distinguish between trajectories that pass a hazard and those that do not. This is the case for both latent methods working with the cost: the method using the cost weight in the contrastive loss and the one using the additional cost network. In Figure IV.18, we can see that the algorithm is able to reach the goal and separate the trajectories into different modes, but not to distinguish these modes based on their cost. One possible reason for this is that weighting the transitions according to their cost is not sufficient to encode meaningful differences. The resulting signal may be too weak to have a meaningful impact on the embedding network. Original DDiffPG is also unable to separate trajectories based on their cost and, in most cases, does not provide meaningful cluster assignments. Addressing this issue with the cost signal is therefore left for future work.

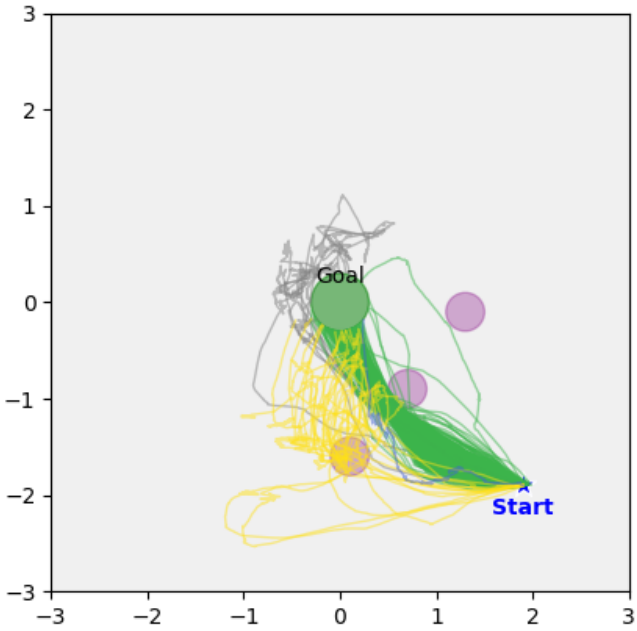


Figure IV.18.: Behavior modes discovered by Latent DDiffPG CTEC COST on Safety Gymnasium Hazards (run 4).

6. Discussion

Overall, the results show that Latent DDiffPG has difficulty preserving the different behavior modes. In AntMaze environments, the algorithm can successfully identify different modes; however, these modes are often merged or discarded as training progresses. As discussed in the previous section, this behavior may be explained by the changing embedding dynamics and our choice of threshold. Similar behavior is observed in the Safety Gymnasium environments. Importantly, this limitation is not unique to the latent extension, as the original DDiffPG also struggles to maintain multiple goal-reaching modes. This suggests that the observed mode collapse may relate to the underlying clustering mechanism or its hyperparameters rather than solely to the choice of the state representation.

Furthermore, the results indicate that the choice of the embedding network has substantial influence on the quality of the discovered clusters. CTEC provides better reconstruction of the coordinates than ICM and EMI and produces a latent space in which the trajectories retain a structure that corresponds to their movement through the environment. EMI may preserve more of how the ant's joints move, which could explain the rather circular structure of a trajectory in the embedding space, while CTEC aims to preserve more of the temporal structure, helping the latent space to keep information about progression through the environment. This finding is also reflected in the online experiments, where CTEC comes closest to producing and preserving multiple meaningful behavior modes in some runs. However, CTEC is still unable to consistently preserve these modes in all environments and seeds. Thus, while the CTEC-based extension shows promising results for behavior mode discovery, it does not fully overcome the observed mode discovery and preservation limitations.

Chapter V.

Limitations, Future Work & Conclusion

1. Limitations

1.1. Clustering Threshold

One major limitation is the sensitivity of clustering to the chosen threshold. In the original algorithm, the authors decide to use tuned thresholds for some of the environments to ensure stable clustering. Our experiments show that the dynamic threshold available in the original algorithm, which calculates the threshold based on the actual linkage distances, also fails to clearly separate different trajectories based on their distances. Furthermore, when using different latent dimensions or training in different environments, a single fixed threshold is unlikely to be suitable across all settings. Therefore, the choice of the clustering method, together with its threshold, can have a major impact on the resulting cluster assignments. The sensitivity makes it difficult to determine whether the difference in the clustering performance results from the quality of the state representation or the clustering configurations themselves. One possible solution could be to use a dynamic, normalized threshold for a suitable embedding rather than restricting the maximum number of clusters, as we do in our experiments.

1.2. Embedding Network

Choosing a suitable embedding network also presents a challenge. It is important to find a network that preserves the information required for trajectory clustering from the original state. In our experiments, we observe that the ICM embedding network is unable to encode the information needed to find distinct behavior modes.

Another challenge arises from continuous updates of the embeddings during training. At the beginning of the training, the embedding network is essentially

random and therefore does not produce useful state embeddings. This can influence the early clustering process and can lead to undesirable cluster assignments in the early stages of training.

1.3. Computational Cost of DTW

Using DTW to calculate the distance between the trajectories can greatly slow down the training when switching to the latent extension. In the original DDiffPG algorithm, DTW is very useful because of its ability to compare trajectories with unequal length. In the original setting, each state has a dimension of two, making DTW calculations relatively inexpensive. However, when switching to the latent extension, DTW needs to operate on trajectories consisting of latent states with increased dimensionality. To encode enough information, latent embeddings sometimes require larger dimensions, which can significantly increase the computational runtime cost of DTW. For our experiments, we increase the latent space dimension to 64. On average, each of the ten runs requires 67 hours to complete training on AntMaze-v1, compared to around six hours for the original algorithm to train for the same number of steps.

Another issue arises from continuous updates of the embedding network. After every embedding update, the previously calculated trajectory distances can no longer be used, as they are computed in the outdated latent space. Since the updated embeddings change the representations of the states, the old distances do not reflect the current trajectory distances anymore and therefore need to be discarded. As a result, the latent extension requires additional distance calculations throughout the training.

2. Future Work

2.1. Cost Implementation and Clustering Threshold

Due to time constraints, we are not able to ablate different approaches for incorporating the cost signal into the embedding. However, switching to the latent extension offers the possibility of including additional important information, such as the cost, in the state representation. Therefore, it is a goal for future work following this thesis to further explore suitable approaches for cost inclusion.

At the moment, the cost is only incorporated on a transition basis by associating the cost with one particular state, thereby losing information about the cost over the entire trajectory. The idea of incorporating the cost on a trajectory basis can therefore be approached together with the trajectory clustering approach explained in Section V.2.2.

The same applies to the clustering threshold, which is beyond the scope of this thesis. During the experiments, we found that this offers several opportunities for improvement. Building on this thesis, it would be interesting to explore different approaches for selecting a threshold and improving the clustering method to make clustering in new environments more stable and to identify more distinct behavior modes.

2.2. Trajectory Clustering

Having successfully implemented the latent clustering approach using an embedding network, we plan to extend the project with trajectory clustering. The paper 'Trajectory Clustering: A Partition-and-Group Framework' by Lee et al. (2007) offers an interesting approach worth trying in the future. Incorporating trajectory-level clustering could identify additional patterns and provide safety-relevant information that is not observable at the state-level. It would also eliminate the additional computational overhead introduced by the increased latent space dimensionality when using DTW.

TRACLUS The paper by Lee et al. (2007) presents a method to group trajectories based on common subtrajectories. The idea is that the trajectories should belong to the same cluster if parts of them show similar behavior, even if they diverge after some point. To achieve this, Lee et al. (2007) divide their method into two stages: partitioning and grouping. First, they partition the trajectories into line segments by identifying characteristic points, which are points where the trajectory's behavior changes rapidly. The trajectory is then partitioned into line segments between two consecutive characteristic points. The partitioning step optimizes two properties, namely preciseness and conciseness. Preciseness states that the difference between the trajectory and its set of trajectory partitions should be as small as possible, while conciseness means that the number of trajectory partitions should be minimal. In the clustering stage, the resulting line segments are grouped using a density-based clustering method. A representative segment is then selected for each cluster.

3. Conclusion

The original DDiffPG algorithm is not universally applicable in different environments due to its reliance on (x, y)-coordinates for trajectory clustering. To overcome this limitation, we introduced a latent embedding network that transforms the entire state into a latent representation, which can then be used to calculate trajectory distances for clustering without relying on hand-picked coordinate features. Among the embedding networks evaluated, CTEC showed the

most promising results, performing at least as well as the original algorithm in terms of mode separation and goal-reaching success across the Safety Gymnasium environments. This demonstrates that the Latent DDiffPG algorithm offers the flexibility of being deployed in environments where 2D coordinates are not readily accessible or meaningful, making DDiffPG more readily applicable across domains. However, mode preservation over time remained challenging for both the latent extension and the original algorithm, indicating that the clustering mechanism itself, not only the choice of state representation, still limits the ability to reliably discover and retain multiple behavior modes. While this thesis surfaces several important insights discussed in the Limitations section, we also outline promising directions for future research in the preceding chapter.

Bibliography

Burda, Yuri et al. (2018). "Exploration by random network distillation". In: *arXiv preprint arXiv:1810.12894*.

Haarnoja, Tuomas et al. (2018). "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR, pp. 1861–1870.

Ho, Jonathan, Ajay Jain, and Pieter Abbeel (2020). "Denoising diffusion probabilistic models". In: *Advances in neural information processing systems* 33, pp. 6840–6851.

Huang, Zhiao et al. (2023). "Reparameterized policy learning for multimodal trajectory optimization". In: *International Conference on Machine Learning*. PMLR, pp. 13957–13975.

Ji, Jiaming et al. (2023). "Safety gymnasium: A unified safe reinforcement learning benchmark". In: *Advances in Neural Information Processing Systems* 36, pp. 18964–18993.

Kim, Hyoungseok et al. (2018). "Emi: Exploration with mutual information". In: *arXiv preprint arXiv:1810.01176*.

Kiran, B Ravi et al. (2021). "Deep reinforcement learning for autonomous driving: A survey". In: *IEEE transactions on intelligent transportation systems* 23.6, pp. 4909–4926.

Kober, Jens, J Andrew Bagnell, and Jan Peters (2013). "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11, pp. 1238–1274.

Kushwaha, Ankita et al. (2025). "A survey of safe reinforcement learning and constrained mdps: A technical survey on single-agent and multi-agent safety". In: *arXiv preprint arXiv:2505.17342*.

Ladosz, Pawel et al. (2022). "Exploration in deep reinforcement learning: A survey". In: *Information Fusion* 85, pp. 1–22.

Lee, Jae-Gil, Jiawei Han, and Kyu-Young Whang (2007). "Trajectory clustering: a partition-and-group framework". In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pp. 593–604.

Li, Zechu et al. (2024). "Learning multimodal behaviors from scratch with diffusion policy gradient". In: *Advances in Neural Information Processing Systems* 37, pp. 38456–38479.

Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pp. 529–533.

Mnih, Volodymyr et al. (2016). "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. PMLR, pp. 1928–1937.

Mohamed, Faisal et al. (2026). "Temporal representations for exploration: Learning complex exploratory behavior without extrinsic rewards". In: *arXiv preprint arXiv:2603.02008*.

Pathak, Deepak et al. (2017). "Curiosity-driven exploration by self-supervised prediction". In: *International conference on machine learning*. PMLR, pp. 2778–2787.

Sakoe, Hiroaki and Seibi Chiba (1978). "Dynamic programming algorithm optimization for spoken word recognition". In: *IEEE transactions on acoustics, speech, and signal processing* 26.1, pp. 43–49.

Schulman, John et al. (2017). "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347*.

Silver, David et al. (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419, pp. 1140–1144.

Sohl-Dickstein, Jascha et al. (2015). "Deep unsupervised learning using nonequilibrium thermodynamics". In: *International conference on machine learning*. PMLR, pp. 2256–2265.

Tang, Haoran and Tuomas Haarnoja (Oct. 2017). *Learning Diverse Skills via Maximum Entropy Deep Reinforcement Learning*. Berkeley Artificial Intelligence Research Blog. URL: `https://bair.berkeley.edu/blog/2017/10/06/soft-q-learning/`.

Wang, Zhendong, Jonathan J Hunt, and Mingyuan Zhou (2022). "Diffusion policies as an expressive policy class for offline reinforcement learning". In: *arXiv preprint arXiv:2208.06193*.

Watkins, Christopher JCH and Peter Dayan (1992). "Q-learning". In: *Machine learning* 8.3, pp. 279–292.

Weng, Lilian (2021). "What are diffusion models?" In: *lilianweng. github. io*, p. 21.

Williams, Ronald J (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3, pp. 229–256.

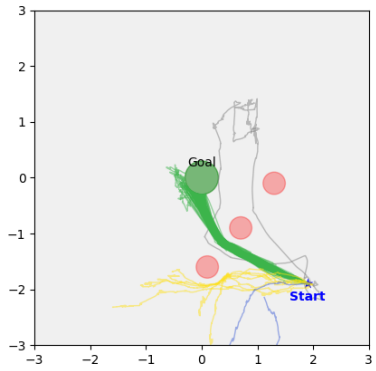
Yang, Long et al. (2023). "Policy representation via diffusion probability model for reinforcement learning". In: *arXiv preprint arXiv:2305.13122*.

Appendix A.

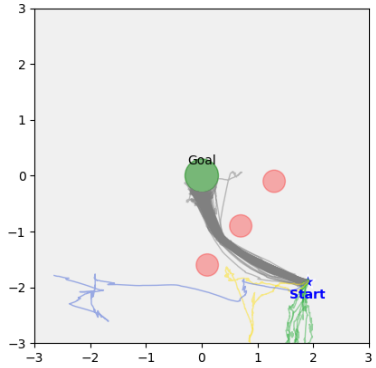
Additional Experiment Results

The figures in the following appendix show additional mode assignment results across different training seeds, providing further support for the observations and claims made in Sections IV.3 and IV.4.

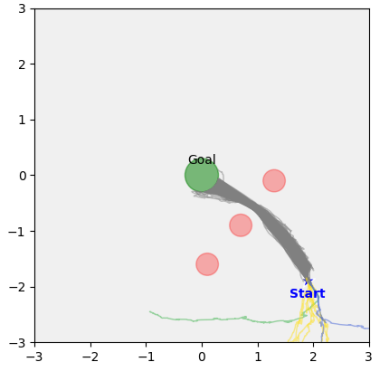
1. Behavior Modes Discovered by DDiffPG on Safety Gymnasium Pillars



(a) 1.5 million steps (run 1)



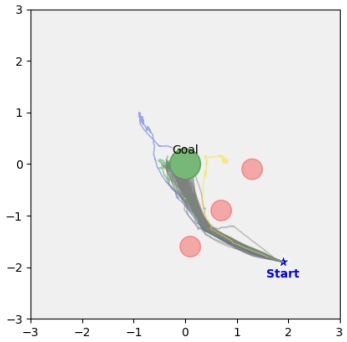
(b) 1.5 million steps (run 2)



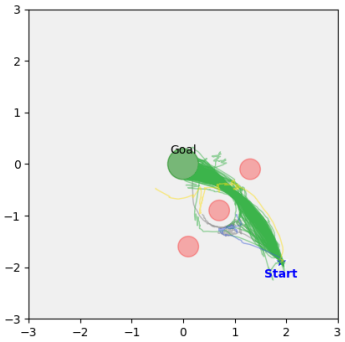
(c) 1.5 million steps (run 7)

Figure A.1.: Behavior modes discovered by DDiffPG in three additional training runs.

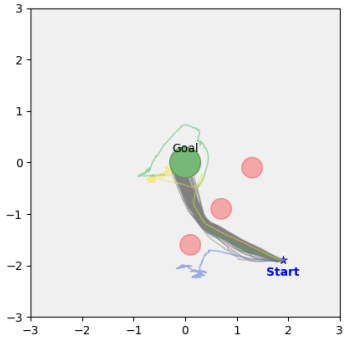
2. Behavior Modes Discovered by Latent DDiffPG on Safety Gymnasium Pillars



(a) 1.5 million steps (run 3)

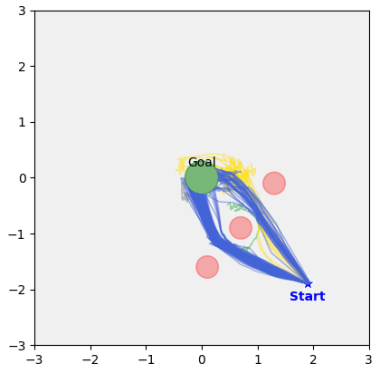


(b) 1.5 million steps (run 7)

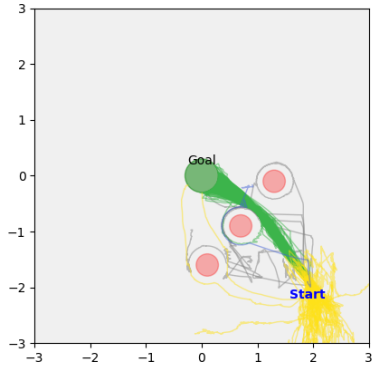


(c) 1.5 million steps (run 10)

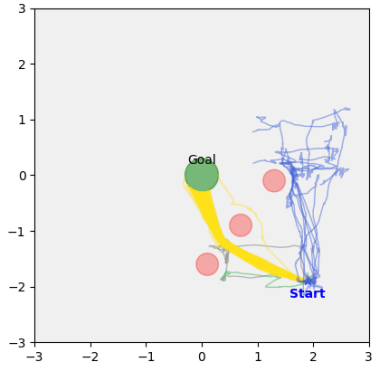
Figure A.2.: Behavior modes discovered by Latent DDiffPG ICM in three additional training runs.



(a) 1.5 million steps (run 1)

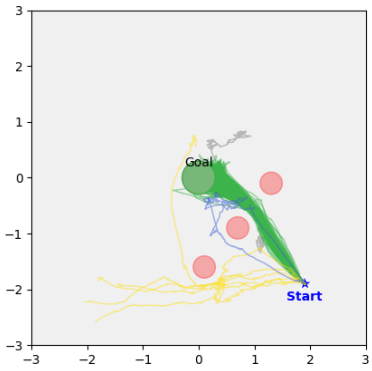


(b) 1.5 million steps (run 7)

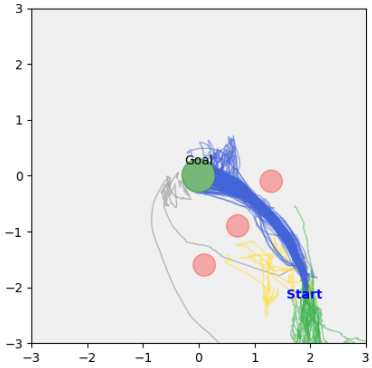


(c) 1.5 million steps (run 10)

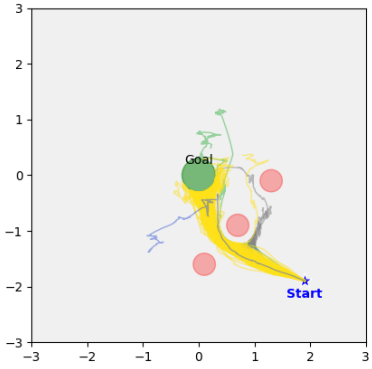
Figure A.3.: Behavior modes discovered by Latent DDiffPG EMI in three additional training runs.



(a) 1.5 million steps (run 2)



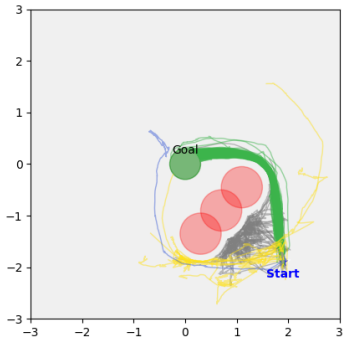
(b) 1.5 million steps (run 3)



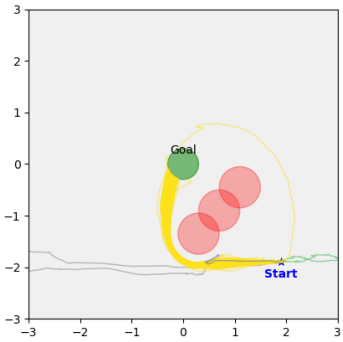
(c) 1.5 million steps (run 5)

Figure A.4.: Behavior modes discovered by Latent DDiffPG CTEC in three additional training runs.

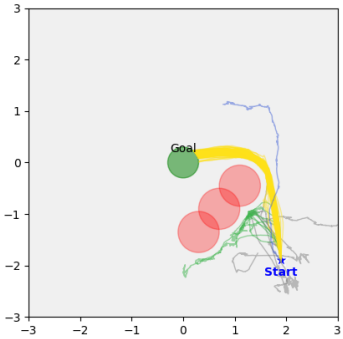
3. Behavior Modes Discovered by Latent DDiffPG on Safety Gymnasium Wall



(a) 1.5 million steps (run 3)



(b) 1.5 million steps (run 5)



(c) 1.5 million steps (run 6)

Figure A.5.: Behavior modes discovered by Latent DDiffPG CTEC in three additional training runs.