

# Toward Fast and Generalizable Decision-Making with Diffusion Models

Xintong Duan

CMU-RI-TR-25-74

July 21, 2025



The Robotics Institute  
School of Computer Science  
Carnegie Mellon University  
Pittsburgh, PA

**Thesis Committee:**

Jeff Schneider, *chair*

Guanya Shi

Mihir Prabhudesai

*Submitted in partial fulfillment of the requirements  
for the degree of Master of Science in Robotics.*

Copyright © 2025 Xintong Duan. All rights reserved.



*To my family.*



## Abstract

Many real-world decision-making problems are combinatorial in nature, where states (e.g., surrounding traffic of a self-driving car) can be seen as a combination of basic elements (e.g., pedestrians and trees). Due to combinatorial complexity, observing all combinations of basic elements in the training set is infeasible, which leads to an essential yet understudied problem of *zero-shot generalization to states that are unseen combinations of previously seen elements*. In this work, we first formalize this problem and then demonstrate how existing value-based reinforcement learning (RL) algorithms struggle due to unreliable value predictions in unseen states. We argue that this problem cannot be addressed with exploration alone, but requires more expressive and generalizable models. We demonstrate that behavior cloning with a conditioned diffusion model trained on expert trajectories generalizes better to states formed by new combinations of seen elements than traditional RL methods. Through experiments in maze, driving, and multiagent environments, we show that conditioned diffusion models outperform traditional RL techniques and highlight the broad applicability of our problem formulation.

Although diffusion models have achieved strong generalization in decision-making tasks, their slow inference speed remains a key limitation. While the consistency model offers a potential solution, its applications to decision-making often struggle with suboptimal demonstrations or rely on complex concurrent training of multiple networks. In this work, we propose a novel approach to consistency distillation for offline reinforcement learning that directly incorporates reward optimization into the distillation process. Our method enables single-step generation while maintaining higher performance and simpler training. Empirical evaluations on the Gym MuJoCo benchmarks and long horizon planning demonstrate that our approach can achieve an 8.7% improvement over previous state-of-the-art while offering up to  $142\times$  speedup over diffusion counterparts in inference time.

Together, this thesis presents diffusion models as a fast and generalizable model for decision-making tasks.



## Acknowledgments

I would like to express my deepest gratitude to my advisor, Professor Jeff Schneider, for his constant support, insightful guidance, and invaluable feedback throughout my research journey. I am also sincerely thankful to my thesis committee, Professor Guanya Shi and Mihir Prabhudesai for their valuable time and thoughtful feedback. I want to thank my collaborators, Yutong He, Fahim Tajwar, and Wenze Chen, for their inspirational discussions, which have helped shape many of the ideas in this thesis. This work would not have been possible without their guidance and support. Finally, I am profoundly grateful to my family and friends for their constant support, patience, and encouragement.



## **Funding**

This work was supported by DSTA.



# Contents



|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>1</b>  |
| <b>2</b> | <b>State Combinatorial Generalization In Decision Making With Conditional Diffusion Models</b>          | <b>3</b>  |
| 2.1      | Motivation . . . . .  | 3         |
| 2.2      | Problem Formulation . . . . .   | 6         |
| 2.2.1    | States Formed by Element Combinations . . . . .   | 6         |
| 2.2.2    | Generalization on Probability Space Support . . . . .   | 7         |
| 2.2.3    | Constraint for OOC Generalization . . . . .   | 8         |
| 2.3      | Why traditional RL fails . . . . .  | 8         |
| 2.3.1    | RL and expected reward estimation . . . . .   | 8         |
| 2.4      | Why Diffusion Models Can Generate OOC Samples . . . . .   | 10        |
| 2.4.1    | Diffusion Models . . . . .  | 10        |
| 2.4.2    | OOC Generalization in Diffusion Models . . . . .  | 10        |
| 2.5      | Conditioned Planning with Diffusion . . . . .   | 12        |
| 2.5.1    | OOC States Lead to Future OOC States . . . . .  | 12        |
| 2.5.2    | Conditional Diffusion for Trajectory Prediction . . . . .   | 12        |
| 2.6      | Experiments . . . . .   | 13        |
| 2.6.1    | Experiment Setup . . . . .  | 13        |
| 2.6.2    | Single-agent Environment . . . . .  | 13        |
| 2.6.3    | Multi-agent Environment . . . . .   | 15        |
| 2.6.4    | How Do Conditional Diffusion Models Generalize to OOC States? . . . . .                                 | 16        |
| 2.7      | Ablations . . . . .   | 17        |
| 2.7.1    | Necessity of Combinatorial Inductive Bias . . . . .   | 18        |
| 2.7.2    | Model Architecture: Attention vs Concatenation . . . . .  | 19        |
| 2.8      | Conclusion . . . . .  | 19        |
| <b>3</b> | <b>Accelerating Diffusion Models in Offline RL via Reward-Aware Consistency Trajectory Distillation</b> | <b>21</b> |
| 3.1      | Motivation . . . . .  | 21        |
| 3.2      | Background . . . . .  | 23        |
| 3.2.1    | Problem Setting . . . . .   | 23        |
| 3.2.2    | Diffusion Models . . . . .  | 24        |
| 3.2.3    | Consistency Trajectory Distillation . . . . .   | 25        |

|          |  |           |
|----------|--|-----------|
| 3.3      | Method . . . . .   | 27        |
| 3.3.1    | Motivation and Intuition . . . . .   | 27        |
| 3.3.2    | Modeling Action Sequences . . . . .  | 28        |
| 3.3.3    | Reward-Aware Consistency Trajectory Distillation . . . . .                                     | 28        |
| 3.3.4    | Decoupled Training . . . . .   | 29        |
| 3.3.5    | Reward Objective as Mode Selection . . . . .   | 30        |
| 3.4      | Experiment . . . . .   | 31        |
| 3.4.1    | Offline RL . . . . .   | 31        |
| 3.4.2    | Long Horizon Planning . . . . .  | 32        |
| 3.4.3    | Inference Time Comparison . . . . .  | 33        |
| 3.5      | Ablation Study . . . . .   | 34        |
| 3.5.1    | Impact of Reward Objective . . . . .   | 35        |
| 3.5.2    | Effect of Reward Objective Weights . . . . .   | 35        |
| 3.5.3    | Number of Sampling Steps . . . . .   | 36        |
| 3.6      | Related Work . . . . .   | 36        |
| 3.7      | Conclusion . . . . .   | 37        |
| <b>4</b> | <b>Conclusions</b>   | <b>39</b> |
| <b>A</b> | <b>State Combinatorial Generalization In Decision Making With Conditional Diffusion Models</b> | <b>41</b> |
| A.1      | Related Work . . . . .   | 41        |
| A.1.1    | Generalization in RL . . . . .   | 41        |
| A.1.2    | Combinatorial Generalization in Computer Vision . . . . .                                      | 42        |
| A.2      | Proof of Corollary 5.1 . . . . .   | 43        |
| A.3      | Additional Visualization of Value Function of PPO . . . . .                                    | 44        |
| A.4      | Experiment Details . . . . .   | 45        |
| A.4.1    | Hardware and Platform . . . . .  | 45        |
| A.4.2    | Statistics . . . . .   | 45        |
| A.4.3    | Model Architecture . . . . .   | 45        |
| A.4.4    | Trajectory formulation . . . . .   | 46        |
| A.4.5    | Pseudo-code . . . . .  | 46        |
| A.4.6    | Maze2D . . . . .   | 47        |
| A.4.7    | Roundabout . . . . .   | 48        |
| A.4.8    | Setup for Roundabout Environment . . . . .   | 49        |
| A.4.9    | StarCraft . . . . .  | 53        |
| A.4.10   | Model Runtime and GPU Memory . . . . .   | 60        |
| A.4.11   | Parameter Comparison with Concatenation or Attention . . . . .                                 | 61        |
| A.4.12   | Substituting More MAPPO Agents with Diffusion Agents . . . . .                                 | 62        |

|   |           |
|---|-----------|
| <b>B Accelerating Diffusion Models in Offline RL via Reward-Aware Consistency Trajectory Distillation</b> | <b>65</b> |
| B.1 Model Architecture . . . . .  | 65        |
| B.1.1 Model Sizes for Maze2d . . . . .  | 65        |
| B.1.2 Model Sizes for Gym-MuJoCo . . . . .  | 65        |
| B.2 Training Details . . . . .  | 66        |
| B.2.1 Noise Scheduler . . . . .   | 66        |
| B.2.2 Reward Discount Factor . . . . .  | 66        |
| B.2.3 Weight of Different Losses . . . . .  | 67        |
| B.3 More Ablations . . . . .  | 68        |
| B.3.1 walker-medium . . . . .   | 68        |
| B.3.2 Comparing with fast sampling algorithms for Maze2d . . . . .  | 68        |
| B.4 The Trade-off between Mode Selection and Sample Diversity . . . . .                                   | 69        |
| B.5 Limitations and Future work . . . . .   | 69        |
| <b>Bibliography</b>   | <b>71</b> |

*When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.*

# List of Figures

|      |  |    |
|------|--|----|
| 2.1  | Different forms of out-of-distribution states.  are seen base elements and  is unseen base element. Their combination forms the sample space. Classic distribution shift assumes <b>states to have the same support but different probability density</b> . We study generalization for <b>out-of-combination states</b> in this work, where test time state distribution has <b>different</b> and possibly <b>non-overlapping</b> support compared to <b>training states</b> . . . . . | 5  |
| 2.2  | Expected Q value of CQL and actual return-to-go (RTG) in unsupported states in Roundabout environment. . . . .   | 9  |
| 2.3  | Total number of crashes and average reward for BC(MLP), PPO, CQL, and diffusion model in the testing environment. Results of models with comparable sizes are shown in Table A.5 to eliminate the concern of different model sizes. . . . .  | 14 |
| 2.4  | Relative improvement % compared to MAPPO on two SMACv2 scenarios: 3v3 and 5v5. Conditional Diffusion show large improvements over the MAPPO baseline, specially in the hard scenario, where we train on teams with the same unit type only but test on random team compositions. Detailed numbers can be found in Table A.10 and Table A.11. . . . .   | 16 |
| 2.5  | Rendering of future states predicted by the diffusion model given same initial state but different conditionings. The grey box is the current state. Blue backgrounds are conditional on all Squares (long attack range) and pink backgrounds are conditioned on all circles (short attack range). Smaller sizes represent less shield and health. More examples shown in Appendix A.4.9. . . . .  | 17 |
| 2.6  | train traj . . . . .   | 18 |
| 2.7  | ID (Uncond) . . . . .  | 18 |
| 2.8  | OOC (Uncond) . . . . .   | 18 |
| 2.9  | cfg=1.3 (Cond) . . . . .   | 18 |
| 2.10 | Trajectories generated in Maze2D for large maze. (a) Samples from the training set. (b) Trajectories directly generated by the unconditioned diffusion model given in distribution start and end positions. (c) Trajectories directly generated by the unconditioned diffusion model on unseen start and end positions. (d) Trajectories directly generated by a conditioned diffusion model using 3 waypoints (black dots) as conditioning with classifier-free guidance (cfg) weight 1.3. For results in medium maze please refer to Appendix A.4.6. . . . .   | 18 |
| 2.11 | Improvement percentage over MAPPO for different types of conditioning in SMACv2. . . . .   | 19 |

|      |   |    |
|------|---|----|
| 3.1  | Overview of Reward Aware Consistency Trajectory Distillation (RACTD).<br>We incorporate reward guidance with consistency trajectory distillation<br>to train a student model that can generate actions with high rewards<br>with only one denoising step. . . . .   | 23 |
| 3.2  | Visualization of CTM loss, DSM loss and reward loss. . . . .  | 26 |
| 3.3  | The reward distribution of the D4RL hopper-medium-expert dataset<br>and 100 rollouts from an unconditioned teacher, an unconditioned<br>student, and RACTD. . . . .   | 30 |
| 3.4  | Ablation on reward objective weight. . . . .  | 34 |
| A.1  | Value prediction of PPO and actual return-to-go (RTG) in unsupported states in<br>Roundabout environment . . . . .  | 44 |
| A.2  | Model architecture. . . . .   | 46 |
| A.3  | train traj. . . . .   | 48 |
| A.4  | ID (Unconditioned) . . . . .  | 48 |
| A.5  | OOD (Unconditioned) . . . . .   | 48 |
| A.6  | cfg=1.3 (Conditioned) . . . . .   | 48 |
| A.7  | Trajectories generated in Maze2D for medium maze. (a) are samples<br>from the training set. (b) are trajectories generated by the uncondi-<br>tioned diffusion model given in distribution start and end positions. (c)<br>are generated by the unconditioned diffusion model on unseen start and<br>end positions. (d) are generated by a conditioned diffusion model using<br>3 waypoints (black dots) as conditioning with classifier-free guidance<br>(cfg) weight 1.3. . . . . | 48 |
| A.13 | Distribution of initial state for 3v3 simple scenario . . . . .   | 56 |
| A.14 | Distribution of initial state for 5v5 simple scenario . . . . .   | 56 |
| A.15 | Rendering of future states predicted by the diffusion model given<br>different conditionings. The grey box is the initial state. Yellow boxes<br>are conditioned on the type of unit in the initial state. Green boxes<br>are conditioned on all Triangles. Smaller sizes represent less shield or<br>health. . . . .   | 59 |
| A.16 | Success rate vs number of agents in SMACv2 5v5 hard scenario that are replaced<br>with diffusion agents. Replacing more than one MAPPO agent with diffusion<br>agents hurts performance. . . . .  | 62 |
| A.8  | train env 1 (cars) . . . . .  | 63 |
| A.9  | train env 2 (bikes) . . . . .   | 63 |
| A.10 | test env 1 . . . . .  | 63 |
| A.11 | test env 2 . . . . .  | 63 |

|   |    |
|---|----|
| A.12 Training and testing environments for Roundabout. The green vehicle is the ego agent and the blue ones are controlled by the environment. The large blue box represents a car and the small blue box represents a bicycle. . . . . | 63 |
|---|----|

# List of Tables

|      |  |    |
|------|--|----|
| 3.1  | (Offline RL) Performance of RACTD and a variety of baselines on the D4RL Gym-MuJoCo benchmark. The best score is emphasized in bold and the second-best is underlined. . . . .   | 32 |
| 3.2  | (Long-horizon planning) The performance of RACTD, Diffuser, and prior model-free algorithms in the Maze2D environment. The best score is in bold and the second-best is underlined. . . . .  | 33 |
| 3.3  | Wall clock time and NFEs per action for different samplers and Diffuser on MuJoCo hopper-medium-replay. . . . .  | 34 |
| 3.4  | We compare incorporating the reward model in different stages of training on MuJoCo hopper-medium-replay. Results are presented as the mean and standard error across 100 seeds. . . . .   | 35 |
| 3.5  | Inference time, NFE, and score comparison for student model multi-step sampling on MuJoCo hopper-medium-replay. . . . .  | 35 |
| A.1  | Training parameter for diffusion model in Maze2D . . . . .   | 49 |
| A.2  | Parameters for car and bicycles in Roundabout environment . . . . .  | 49 |
| A.3  | Training parameter for PPO . . . . .   | 50 |
| A.4  | Training parameter for diffusion model in Roundabout . . . . .   | 51 |
| A.5  | Model size, number of parameters, and performance for different models. . . . .  | 51 |
| A.6  | Performance of conditioned diffusion model given ground truth and random conditionings. . . . .  | 52 |
| A.7  | MAPPO hyper-parameters used for SMACv2. We utilize the hyperparameters used in SMACv2 [17]. . . . .  | 53 |
| A.8  | Training parameter for diffusion model in StarCraft for 5v5 . . . . .  | 54 |
| A.9  | Training parameter for diffusion model in StarCraft for 3v3 . . . . .  | 55 |
| A.10 | Success rate of each agent in 100 rounds. The first two rows correspond to the simple setting of generalization to states with different support and the last two rows correspond to non-overlapping support. Numbers in the parenthesis represent the standard error over 3 seeds. The best performing method is labeled bold. The 2 PPO + 1 Rand column shows the effect of replacing one PPO trained agent with a random agent as a baseline for comparison against the 2 PPO + 1 Diffusion case. . . . . | 57 |

|      |  |    |
|------|--|----|
| A.11 | Success rate of each agent in 100 rounds. The first two rows correspond to the simple setting of generalization to states with different support and the last two rows correspond to non-overlapping support. Numbers in the parenthesis represent the standard error over 3 seeds. The best performing method is labeled bold. The 4 PPO + 1 Rand column shows the effect of replacing one PPO trained agent with a random agent as a baseline for comparison against the 4 PPO + 1 Diffusion case. | 57 |
| A.12 | Ablation for Diffusion on 3v3 . . . . .  | 58 |
| A.13 | Ablation for Diffusion on 5v5 . . . . .  | 58 |
| A.14 | SMAC II success rate for 2v2 . . . . .   | 59 |
| A.15 | Training time for PPO and conditioned diffusion model in different environments. . . . .   | 60 |
| A.16 | GPU Memory for training conditioned diffusion model in different environments. . . . .   | 60 |
| A.17 | Number of parameters in attention-based conditioning and concatenation-based conditioning. . . . .   | 61 |
| B.1  | Model parameters for Unet in Maze2d. . . . .   | 66 |
| B.2  | Model parameters for Unet in MuJoCo. . . . .   | 66 |
| B.3  | Reward model parameters in MuJoCo. . . . .   | 67 |
| B.4  | Weights for CTM, DSM, and Reward loss used in MuJoCo benchmark.  | 67 |
| B.5  | We compare incorporating the reward model in different stages of training on MuJoCo walker-medium. Results are presented as the mean and standard error across 100 seeds. . . . .  | 68 |
| B.6  | We compare fast sampling algorithms DDIM and CTD, along with our method RACTD on Maze2d environment. DDIM performs fast sampling based on a DDPM model, while CTD and RACTD (ours) distill an EDM teacher. The number of function evaluations (NFE) reflects the sampling speed of each algorithm. Results are reported as the mean and standard error over 100 random seeds. . . . .  | 68 |
| B.7  | A summarization of the trade off between sample diversity and model performance. . . . .   | 69 |

# Chapter 1

## Introduction

Real-world decision-making tasks often involve environments composed of a combinatorial set of fundamental elements—such as pedestrians, vehicles, and road features in driving scenarios. This combinatorial nature poses a significant challenge for generalization: it is impractical to observe all possible combinations during training. Humans overcome this by learning how to interact with individual elements and extrapolating this knowledge to unseen configurations. Inspired by this, we aim to develop algorithms that generalize to *unseen combinations of known elements*, a capability we refer to as out-of-combination (OOC) generalization.

Despite the remarkable success of reinforcement learning (RL) across various domains, many RL algorithms, especially in offline settings, struggle under distributional shifts where testing scenarios include novel state compositions not encountered during training. Prior work has largely overlooked this specific form of generalization, instead focusing either on in-support distribution shifts or settings that assume access to unseen elements, which makes the problem ill-posed. In contrast, we formalize and tackle the OOC generalization problem, offering a setting that is both realistic and analytically tractable.

In Chapter 2, we propose leveraging diffusion models, which are naturally suited for capturing compositional structure and modeling complex state distributions to address this challenge. We show that a conditioned diffusion model can serve as both a world model and a planner, enabling robust zero-shot generalization in OOC scenarios across diverse environments, including maze navigation, driving simulations,

and multi-agent games.

While diffusion models offer strong expressivity and generalization capabilities, they suffer from high computational overhead due to their iterative sampling procedures. This limitation hinders their practical deployment in time-sensitive or large-scale applications. To address this limitation, recent work in image generation proposed consistency distillation [76], which distills a multi-step diffusion teacher into a single-step student model while maintaining generation quality.

In Chapter 3, we introduce how to extend consistency trajectory distillation [41] to offline RL tasks, by explicitly incorporating a reward optimization objective during the student distillation process. This augmentation encourages the student consistency model not only to mimic the diverse behaviors of the teacher diffusion model but also to prefer high-reward trajectories identified by the reward model. Unlike actor-critic based diffusion methods, our approach decouples the reward model training from the distillation process, simplifying training while enhancing performance.

Through comprehensive evaluations, we demonstrate that our approach achieves superior performance and significant acceleration in sampling on offline RL benchmarks. In particular, our method outperforms state-of-the-art offline RL algorithms on challenging long-horizon and multi-modal benchmarks, achieving an 8.7% performance gain and up to  $142\times$  speedup in inference.

Together, this thesis contributes to the development of diffusion models that combine expressiveness for generalization to novel scenarios with the efficiency required for real-world deployment, supporting fast and generalizable decision-making.

# Chapter 2

## State Combinatorial Generalization In Decision Making With Conditional Diffusion Models

### 2.1 Motivation

In many real-world decision-making tasks, environments can be broken down into combinations of fundamental elements. For instance, in self-driving tasks, the surrounding environment consists of elements like bicycles, pedestrians, and cars. Due to the exponential growth of possible element combinations, it is impractical to encounter and learn from every possible configuration during training. Rather than learning how to act in each unique combination, humans instead learn to interact with individual elements – such as following a car or avoiding pedestrians – and then extrapolate this knowledge to unseen combinations of elements. Therefore, it is important to study the *generalization to unseen combinations of known elements*, hereafter referred to as the out-of-combination (OOC) generalization, and to develop algorithms that can effectively handle these unseen scenarios.

Despite the success of reinforcement learning (RL) in decision-making tasks, many existing RL algorithms, particularly in offline settings, struggle to perform adeptly under state distribution shifts between training and testing, which typically occur

when the learned policy visits states that differ from the data collection policy at test time [37, 46, 55, 71]. While there have been works studying this problem, most of them either (1) focus on distribution shifts where the training and testing sets share the same support but different probability densities, without accounting for the presence of entirely new and unseen element combinations [19, 21], or (2) allow unseen elements in test combinations, which makes the problem ill-posed without introducing other potentially unrealistic assumptions [77, 87]. As a result, these works have failed to recognize and address the critical challenge of generalization to unseen combinations of seen elements and therefore fail to capture and compose existing knowledge for these fundamental elements.

In this work, we directly tackle the problem of state combinatorial generalization in decision-making tasks, where testing states consist of unseen combinations of elements encountered during training. As illustrated in Figure 2.1, our task differs conceptually from traditional distribution shift problems. Unlike simple distribution shifts, where the testing set remains within the support of the training set, our proposed task requires algorithms to handle out-of-support states that are never seen during training. This makes our problem both more challenging and more representative of real-world scenarios. At the same time, our OOC setting is better defined than the unconstrained out-of-support (OOS) setting, where testing states may include completely arbitrary unseen elements and therefore is inadequately formulated and intractable without other potentially impractical assumptions such as the existence of state distance metrics [77] or isomorphic Markov decision processes (MDPs) [87]. By focusing on new combinations of known elements, our setting strikes a balance between real-world applicability and tractability, making it more suitable for standardized evaluation and formal analysis.

To facilitate this study, we first provide formal definitions of state combination and OOC generalization. We then demonstrate the challenge of this task by showing how traditional RL algorithms struggle to generalize in this setting due to unreliable value prediction, and the need for a more expressive policy. On the hunt for a suitable solution, we draw inspiration from the linear manifold hypothesis in diffusion models [10, 27] and recent advances in combinatorial image generation [62], and present diffusion models as a promising direction by showing how they can naturally account for the combinatorial structure of states into the diffusion process, enabling

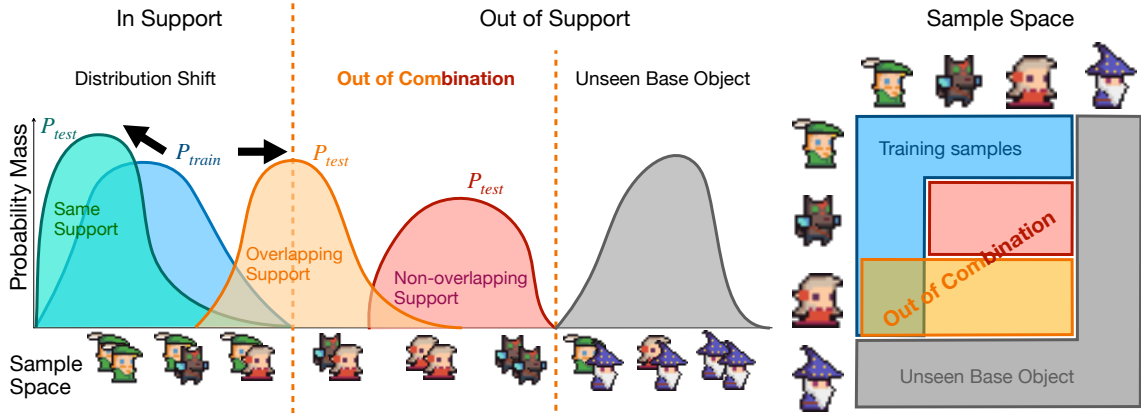






Figure 2.1: Different forms of out-of-distribution states.    are seen base elements and  is unseen base element. Their combination forms the sample space. Classic distribution shift assumes states to have the same support but different probability density. We study generalization for **out-of-combination** states in this work, where test time state distribution has **different** and possibly **non-overlapping** support compared to **training states**.

them to sample OOC states. Furthermore, we analyze how this feature enhances world model predictions and propose leveraging a conditioned diffusion model to simultaneously serve as both a world model and a planner for this task.

Experimentally, we evaluate the models on three distinctly different RL environments: maze, driving, and multiagent games. All three settings are easily adaptable to the OOC generalization problem using existing RL frameworks, demonstrating the broad applicability of the combinatorial state setup. We demonstrate behavior cloning (BC) with a conditioned diffusion model outperforms not only vanilla BC and offline RL methods like CQL [43] but also online RL methods like PPO [72] in zero-shot OOC generalization. To explore factors contributing to its generalization, we visualize the states predicted by the conditioned diffusion model. Our results demonstrate that the model effectively captures the core attributes of each base element and accurately composes future states by integrating these fundamental attributes. We demonstrate that, while exploration is commonly used to enhance model generalization, zero-shot OOC generalization relies instead on the use of a more expressive policy.

## 2.2 Problem Formulation

In this section, we formally define the problem of state combinatorial generalization by providing definitions for state combination and identify out-of-combination generalization as a problem for generalization to different supports in the same sample space.

### 2.2.1 States Formed by Element Combinations

Following Wiedemer et al. [82], we first denote  $e \in \mathbf{E}$  to be a *base element* for an environment. A base element is defined to be the most elementary and identifiable element that is relevant to the decision making task of interest. For example, in a traffic environment, the set  $\mathbf{E}$  can be the set of vehicles that can occur in the environment such as {car, bike}; and in a 2D maze environment, the set  $\mathbf{E}$  can be the set of possible locations labeled by the  $x, y$ -axis coordinate of the agent, i.e.  $\mathbb{R}^2$ . Suppose there are a finite number of  $n$  base elements in an environment. Since these elements are the fundamental components relevant to the decision making task, we can form a *latent vector*  $\mathbf{z} = (z_1, z_2, \dots, z_n) \in \mathbf{Z} \equiv \mathbf{E}^n$ , where  $z_i \in \mathbf{E} \forall i \in \{1, \dots, n\}$  that represents the combination of all rudimentary components appearing in this environment related to the decision making task.

Each element can also be associated with a collection of *attributes*  $\mathbf{r}$  such as the color of the vehicle and the velocity of the agent. Attributes can be used to render the states and the *rendering function*  $f(\mathbf{z}, (\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_n))$  can then map the latent and the attributes to a state  $\mathbf{s} \in \mathbf{S}$ . In the traffic environment example,  $f$  is equivalent to reconstructing the cars and the bikes given their colors and positions, etc. All reconstructed base elements collectively determine a state  $\mathbf{s}$ . Concretely, we provide the following definition:

**Definition 2.2.1** (States and latent vectors). For any state  $\mathbf{s}$  with  $n$  base elements in state space  $\mathbf{S}$  and rendering function  $f$ , we have  $\mathbf{s} = f(\mathbf{z}, (\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_n))$  where the corresponding latent vector  $\mathbf{z}$  in latent space  $\mathbf{Z} \equiv \mathbf{E}^n$  for  $\mathbf{s}$  is  $\mathbf{z} = (z_1, z_2, \dots, z_n)$  where  $z_i \in \mathbf{E}$  for  $i = 1, \dots, n$ .

With our definition of base elements and states, *the combinatorial property of states naturally follows as the composition of different base elements in the latent*

space.

Notice that in practice, for the same environment, one can define different base element sets depending on the desired granularity of the task. In addition, since we usually can only obtain observations of the states, in practice we can only extract the empirical latent vector  $\tilde{\mathbf{z}}$  from the observation.

### 2.2.2 Generalization on Probability Space Support

Since we have identified the fundamental elements of the state in the target decision making task, we can formulate the distribution of states with the probability spaces of latent vectors. When our base element set is discrete, finite, and countable, the probability mass function (PMF)  $p$  can directly ascribe a probability to a sample in  $\mathbf{Z}$ . Then we can define the corresponding probability space as

**Definition 2.2.2** (Probability space for discrete latents). Define the sample space  $\mathbf{Z}$  as the set of all possible  $\mathbf{z}$ .  $\sigma$ -algebra  $\Sigma = 2^{\mathbf{Z}}$  is the power set of  $\mathbf{Z}$ .  $p : \mathbf{Z} \rightarrow [0, 1]$  such that  $\sum_{\mathbf{z} \in \mathbf{Z}} p(\mathbf{z}) = 1$  is the PMF. Then the probability space over the latent vector  $\mathbf{z}$  can be defined as  $P = (\mathbf{Z}, \Sigma, p)$ .

When  $\mathbf{Z}$  is a continuous space, we can also have the corresponding definitions.

**Definition 2.2.3** (Probability space for continuous latents). Define the sample space  $\mathbf{Z}$  as the set of all possible  $\mathbf{z}$ .  $\sigma$ -algebra  $\Sigma = \mathcal{B}(\mathbf{Z})$  is the Borel set of  $\mathbf{Z}$ .  $p : \mathbf{Z} \rightarrow [0, 1]$  such that  $\int_{\mathbf{z} \in \mathbf{Z}} p(\mathbf{z}) d\mathbf{z} = 1$  is the probability dense function (PDF). Then the probability space over the latent vector  $\mathbf{z}$  can be defined as  $P = (\mathbf{Z}, \Sigma, p)$ .

The support of  $P = (\mathbf{Z}, \Sigma, p)$  can then be defined as  $\text{supp } P := \{\mathbf{z} \in \mathbf{Z} : p(\mathbf{z}) > 0\}$ .

State combinatorial generalization, or OOC generalization, is then defined as generalizing to latent probability space with a different support. Denote the latent probability space of training states as  $P_{train} = (\mathbf{Z}, \Sigma_{train}, p_{train})$  and testing states as  $P_{test} = (\mathbf{Z}, \Sigma_{test}, p_{test})$ , then combinatorial generalization assumes  $\text{supp}\{P_{train}\} \neq \text{supp}\{P_{test}\}$ . That is to say, combinatorial generalization in state space requires generalizing to a distribution of latent vectors with different, and possibly non-overlapping support [82]. Whereas traditional distribution shift in RL normally assumes different PMF or PDF ( $p_{train} \neq p_{test}$ ) and generalizing to unseen objects assumes different element set ( $\mathbf{Z}_{train} \neq \mathbf{Z}_{test}$ ) as shown in Figure 2.1.

### 2.2.3 Constraint for OOC Generalization

One crucial assumption made by OOC generalization is that all base elements are seen at training time. Recall  $\mathbf{z} = (z_1, z_2, \dots, z_n)$  where  $z_i \in \mathbf{E}$  for  $i = 1, \dots, n$ . This indicates that the marginal distribution  $p(z_i) > 0$  for all  $z_i$  at training time, or equivalently the training probability space has full support over the marginals. For discrete latent spaces, this also implies that every base element that appeared in the sample space would appear at least once in one latent feature  $\mathbf{z}$ . To ensure full support of base elements, the union of marginal supports at test time should be a subset of that at training time. Finally, to test generalizability, we assume  $\text{supp}\{P_{\text{train}}\} \subsetneq \mathbf{Z}$ , i.e. the training probability space doesn't have full support on the entire latent space.

*Constraint 2.2.4* (Combinatorial support). Given probability spaces  $P = (\mathbf{Z}, \Sigma_P, p)$  and  $Q = (\mathbf{Z}, \Sigma_Q, q)$  over latent vector  $\mathbf{z} = (z_1, z_2, \dots, z_n) \in \mathbf{Z}$  where  $z_i \in \mathbf{E}$  for  $i = 1, \dots, n$ ,  $P$  has full combinatorial support for  $Q$  if:  $\bigcup_{i=0}^n \{z_i \in \mathbf{E} : q(z_i) > 0\} \subseteq \bigcup_{i=0}^n \{z_i \in \mathbf{E} : p(z_i) > 0\}$ .

## 2.3 Why traditional RL fails

Most RL algorithms include estimating the expected cumulative reward of choosing a specific action given the current state [24, 43, 72]. We demonstrate the estimation of value functions is problematic given unsupported states and this theoretically can not be solved by more exploration or more training data in this section.

### 2.3.1 RL and expected reward estimation

Most deep RL algorithms rely on learning a Q or Value function, which takes in the current state as network input and predicts the expected future reward [24, 72]. Since states with unseen composition are unsupported and fall within the undertrained regions of the neural network, the value prediction is highly unreliable. This affects methods that directly choose the maximum action from erroneous Q value predictions and also methods that update the actor with an erroneous value prediction. We plot the expected Q-values learned by CQL alongside the actual return-to-go in both failed

and success scenarios in Roundabout environment [45] (Section 2.6.2) when presented with OOC states in Figure 2.2. The grey dashed line is the expected Q-values the model predicts for in-distribution states.

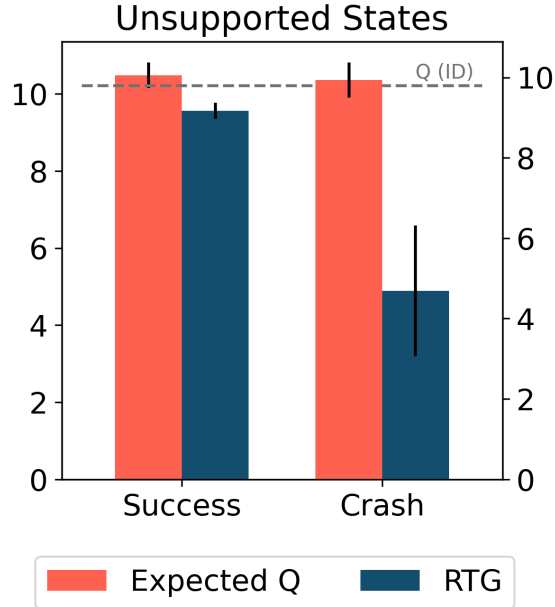


Figure 2.2: Expected Q value of CQL and actual return-to-go (RTG) in unsupported states in Roundabout environment.

One key observation can be made: *Q function shows signs of memorizing, which assigns similar Q values for both training and OOC states.*

Despite the problem of distribution shift being a central challenge for offline RL [46], online methods also suffer from unseen states when zero-shot generalizing to unsupported states. Traditionally distribution shifts are mitigated with a wider training state distribution under the assumption that test time states are sampled from a distribution with different probability densities but the same support. *However, since new states with different object combinations are out of support of the training environment, using a more exploratory online policy or collecting more training trajectories for offline RL will never be able to visit these states and thus will not fundamentally solve this issue.* We need a policy with better generalization to unsupported states to achieve zero-shot generalization in this problem.

## 2.4 Why Diffusion Models Can Generate OOC Samples

We first introduce diffusion model notations and then provide a proof sketch and experimental evidence of why diffusion models are capable of generating OOC samples (states).

### 2.4.1 Diffusion Models

Diffusion models are among the most popular methods for density estimation. Ho et al. [28] proposed DDPM to model the data generation process with a forward and reverse process. In the forward process, noise is added to corrupt data  $\mathbf{x}_t$  iteratively for  $T$  timesteps towards a standard Gaussian distribution.

The target of diffusion modeling is to learn the reverse process  $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$ . This way, we can sample from the data distribution by first obtaining a Gaussian noise  $\mathbf{x}_t$  and then iteratively sampling from  $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ . With reparametrization trick, we can train a model  $\epsilon_\theta$  to predict the noise  $\epsilon$  at each timestep  $t$ , and gradually denoise using update rule  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\alpha_t}-\sigma_t^2} \epsilon_\theta(\mathbf{x}_t, t) \right) + \sigma_t \epsilon_t$ , where  $\epsilon_t \sim \mathcal{N}(0, I)$  with variance schedulers  $\alpha_t, \bar{\alpha}_t$ . Given the same pretrained diffusion model, one can also perform DDIM sampling [73]  $\mathbf{x}_{t-1} = \sqrt{\alpha_{t-1}} \left( \frac{\mathbf{x}_t - \sqrt{1-\alpha_t} \epsilon_\theta(\mathbf{x}_t, t)}{\sqrt{\alpha_t}} \right) + \sqrt{1-\alpha_{t-1}} \epsilon_\theta(\mathbf{x}_t, t) + \sigma_t \epsilon_t$  to enable fast sampling.

Song et al. [75] formally established the connection between diffusion models and score-based stochastic differential equations (SDE). Interestingly, they discovered that each diffusion process has a corresponding probability flow ODE that shares the same intermediate marginal distributions  $p(\mathbf{x}_t, t)$  for all  $t$ . The transformation between probability flow ODE and SDE can be easily achieved by adjusting the random noise hyperparameter  $\sigma$  in DDIM sampling.

### 2.4.2 OOC Generalization in Diffusion Models

We demonstrate that a well-trained diffusion model can naturally sample OOC states that satisfy combinatorial support constraint 2.2.4 with non-zero probability at test

time. The key idea is that pseudo-random denoising trajectories can be constructed at inference time that yield OOC samples with non-zero probability.

Since our states are formed by combinations of base elements (Definition 2.2.1), with well constructed  $\mathbf{Z}$  we can assume that the states lie on a lower dimensional manifold  $\mathcal{M}$  (representing combinations of base elements) embedded in the high dimensional ambient state space. In some cases such as maze navigation where the latent space is a linear subspace, we can even assume that the underlying manifold  $\mathcal{M}$  is a linear manifold whose tangent space is isomorphic to itself. With these assumptions, we present the following corollary whose proof is shown in the appendix A.2.

**Corollary 2.4.1.** *Suppose the states lie along a linear manifold  $\mathcal{M}$  in the state space  $\mathbf{S}$  and the latent space  $\mathbf{Z}$  is well constructed so that  $\mathbf{Z}$  is (affine) isomorphic to  $\mathcal{M}$ . Then a diffusion model  $p_\theta$  that is well trained on  $P_{train}$  can sample an OOC state with non-zero probability.*

While the linear manifold assumption may not hold for more complex states, recent computer vision research provides evidence of the combinatorial generalization capabilities of diffusion models in more complicated data spaces: Okawa et al. [62] showed that given different concepts like shape, color, and size in synthetic shape generation, conditional diffusion models demonstrate a multiplicative emergence of combinatorial abilities where it will first learn how to generalize to concepts closer to the training samples (i.e. only change one of color, shape, and size) and eventually adopt full compositional generalization ability with enough training. Aithal et al. [2] identifies the phenomena where diffusion models generate samples out of the support of training distribution through interpolating different complex modes on a data manifold. Kadkhodaie et al. [36] demonstrate generalization to unsupported data by showing two diffusion models trained on large enough non-overlapping data converge to the same denoising function. In the next sections, we discuss how to use diffusion models to handle this challenging problem and also provide empirical evidence in decision-making tasks.

## 2.5 Conditioned Planning with Diffusion

Recognizing the capability of diffusion models to generate OOC states, we leverage this property by formulating decision-making tasks as conditional state-action generation problems. We begin by demonstrating how OOC initial states can propagate into future OOC states and then present our modeling approach, which capitalizes on this OOC sampling feature to enhance action generation in OOC scenarios.

### 2.5.1 OOC States Lead to Future OOC States

In many real-world scenarios and RL environments, an OOC current state often leads to OOC future states. For instance, in self-driving tasks, if the ego agent is surrounded by two vehicles, it is highly probable that the same vehicles will remain nearby in subsequent timesteps. This phenomenon is even more evident in multi-agent games like StarCraft, where the types of the ego agent’s teammates and enemies remain fixed after the round begins. Such cascading OOC states pose significant challenges for traditional model-based methods, as their accuracy depends on their world model prediction and when these future states are unsupported, they struggle to make accurate future predictions. This limitation motivates the use of conditioned diffusion models as world models, offering a more robust approach to handling OOC scenarios.

### 2.5.2 Conditional Diffusion for Trajectory Prediction

Diffusion models in decision-making tasks can be broadly categorized into two approaches: one focuses on using diffusion models as policy/value network for valued-based RL [38, 80], while the other leverages diffusion models to directly generate future trajectories [3, 34]. The latter approach, which employs diffusion models as both a world model and a planner, aligns more effectively with our goal of addressing OOC state generalization. Following the setup of Janner et al. [34], we train a conditional diffusion model to denoise (predict) future state-action pairs conditioned on the current state and a latent vector. This model is trained via behavior cloning on expert data, capturing the relationship between states, optimal actions, and future states. Consequently, we assume that the actions corresponding to OOC states generated

under new conditions will remain statistically coherent with the underlying dynamics (MDP), facilitating robust generalization to novel scenarios.

## 2.6 Experiments

The primary goal of our experiments is to answer the following questions: (1) (Wide applicability) Does the state-space of different existing RL environments exhibit a compositional nature? (2) (Advantages) What are some interesting features conditional diffusion models have that contribute to their performance when generalizing to OOC states? (3) (Conditioning) Does conditioning help with OOC generalization?

### 2.6.1 Experiment Setup

We test our models in three drastically different existing RL environments: a single-agent driving environment Roundabout, a multiagent cooperative game SMACv2, and a navigation task Maze2d. We use the category of the base elements found in the observation for Roundabout and SMACv2, and the x,y positions of the three waypoints for Maze2d as the latent vector. This information is extracted from observations and is equally accessible to all models. We incorporate a cross-attention layer after each residual block for Diffusion Unet [34] to facilitate learning the correlation between conditioning and trajectory. The expert offline data is collected for behavior cloning by fully trained PPO agents on each combination of the training environment and includes only successful rollouts. The first action in the generated trajectory is then used to step the environment (Appendix A.4.5). Detailed architecture of the model can be found in Appendix A.4.3 and the planning process is described in Algorithm 1.

### 2.6.2 Single-agent Environment

**Environment** HighwayEnv [45] is a self-driving environment where the agent needs to control a vehicle to navigate between traffic controlled by predefined rules. We specifically look at the Roundabout environment with two types of traffic: cars and bicycles (Visualization in Appendix A.4.7).

State in this environment is a composition of four environment vehicles that are either cars or bicycles and the ego agent, which is always a car. Environment observation contains observability, the locations and speed of the ego and surrounding agent, and *whether this agent is a car or a bike (Conditioning)*. During training time, the environment will only generate traffic of all cars or all bicycles with equal probability. During test time, environments will generate a mixture of cars and bicycles (detailed setup in Appendix A.4.8). Cars and bicycles have different sizes, max speeds, and accelerations, leading to different behavior patterns. This is an instance of generalizing to OOC states with non-overlapping support.

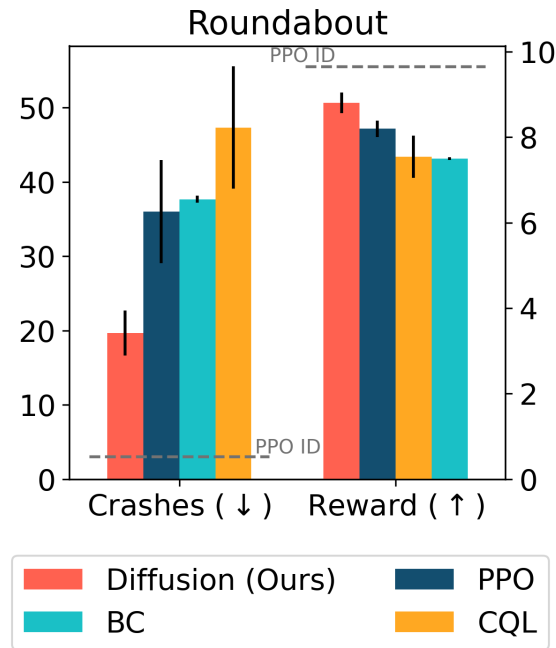


Figure 2.3: Total number of crashes and average reward for BC(MLP), PPO, CQL, and diffusion model in the testing environment. Results of models with comparable sizes are shown in Table A.5 to eliminate the concern of different model sizes.

**Results** As shown in Figure 2.3, conditional diffusion model has almost half the number of crashes and higher reward when zero-shot generalizing to states with mixture traffic. Since we train the diffusion model exclusively on successful PPO trajectories, the training state distribution for diffusion is much narrower compared to PPO, an online method. This is interesting since it is widely acknowledged that

online models have better generalization compared to offline models [46].

**Takeaway 1:** Conditional diffusion models, trained on an offline dataset with narrow state distribution with full combinatorial generalization support, have better zero-shot generalization performance to OOC states compared to online RL trained in the same environment.

### 2.6.3 Multi-agent Environment

**Environment** The StarCraft Multi-Agent Challenge (SMAC/SMACv2) [17, 69] is a multi-agent collaborative game that takes several learning agents, each controlling a single army unit, to defeat the enemy units controlled by the built-in heuristic AI. This benchmark is particularly challenging for its diverse army unit behaviors and complex team combinations, which enable diverse strategies like focus fire and kiting enemies to emerge [18]. Each agent’s observation includes health, shield, position, *unit type (Conditioning)* of its own, visible teammates, and enemies.

We treat one agent as the ego agent, and consider its teammates and enemies as part of the environment. Then states can be naturally seen as compositions of the unit types in a particular playthrough. *We expect the ego agent to generate different policies when playing with or against different types of units, and we aim to test OOC generalization by changing the unit composition in the environment.* Since we use a MAPPO [83] for data collection, we report the performance gain/loss compared to MAPPO. To treat the teammates and enemies of one particular agent as environment and change their combination, we control one unit with a conditional diffusion model and let MAPPO control the rest of its teammates.

**Setup** The unit types in this experiment are Protoss.Stalker, Protoss.Zealot, and Protoss.Colossus, referred to as  $a, b, c$  respectively. We evaluate on two OOC scenarios: (1) (*Simple: Different but overlapping support*): Train the model on randomly generated combinations ( $ABC$ ) of all units and test it where all the units on the team have same type ( $AAA$ ), (2) (*Hard: Non-overlapping support*): the opposite scenario, where we train on teams with only one unit type ( $AAA$ ), but during test-time we see any composition of these three units ( $ABC$ ). More information about our setup could be found in Appendix [A.4.9](#).

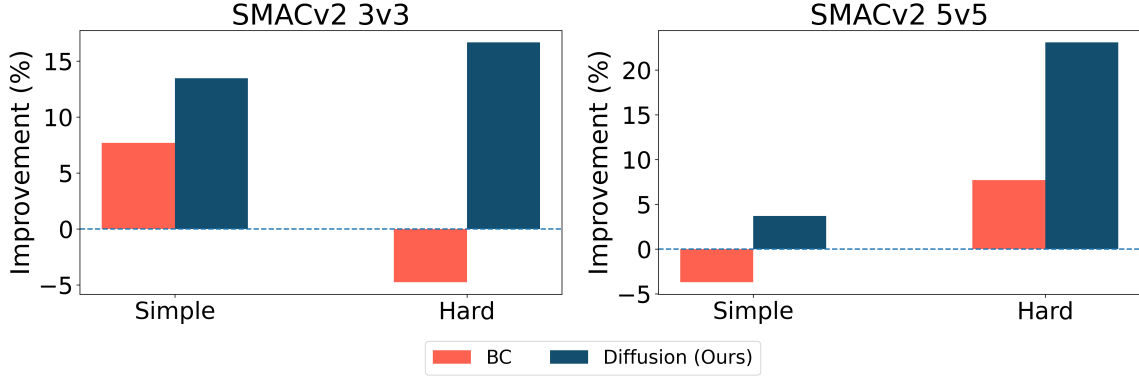


Figure 2.4: Relative improvement % compared to MAPPO on two SMACv2 scenarios: 3v3 and 5v5. Conditional Diffusion show large improvements over the MAPPO baseline, specially in the hard scenario, where we train on teams with the same unit type only but test on random team compositions. Detailed numbers can be found in Table A.10 and Table A.11.

**Results** As shown in Figure 2.4, MAPPO performance drastically dropped in the hard OOC scenario by 55.2% for 5v5 and 50% for 3v3. If we substitute one agent generated by MAPPO with conditional diffusion, the success rate can be improved by 16.7% for 3v3 and 23.1% for 5v5 in hard OOC scenario as shown in Figure 2.4. Detailed success rates are shown in Table A.10 and Table A.11.

**Takeaway 2:** Multi-agent RL, viewed from the perspective of a single ego agent, naturally requires combinatorial generalization to collaborate/compete with different agent types. Compositional complexity can be found in a wide range of distinctly different real-world tasks like driving and multiagent decision-making

## 2.6.4 How Do Conditional Diffusion Models Generalize to OOC States?

To see how diffusion models generalize to OOC states, we render the states predicted by the diffusion models given different conditionings with the same current state, as shown below in Figure 2.5.

We can see that *conditionings determine the unit type of agent predicted by the diffusion model and also their behavior pattern. Whereas the current state determines other attributes like initial location and health.* Different conditionings will lead to different strategies. The circle unit has attack range 1 and the square unit has attack range 6. For units with short attack ranges, the optimal strategy is to approach their

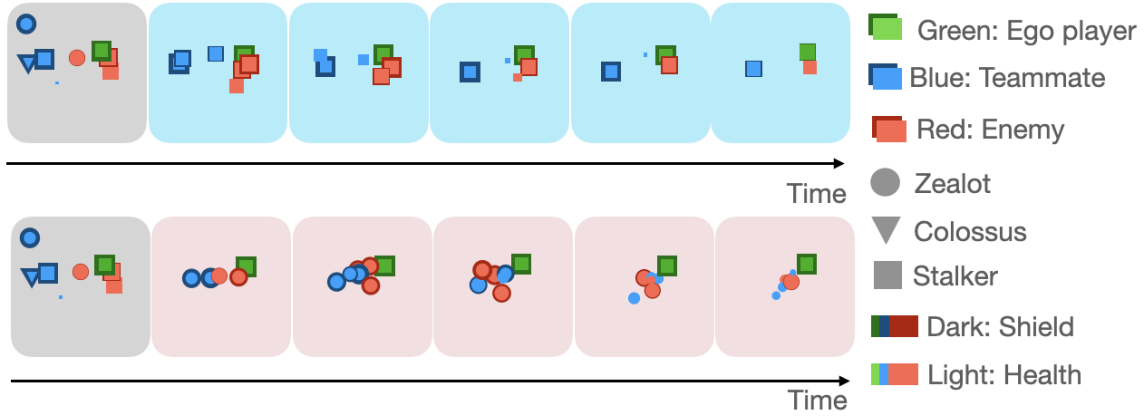


Figure 2.5: Rendering of future states predicted by the diffusion model given same initial state but different conditionings. The grey box is the current state. Blue backgrounds are conditional on all Squares (long attack range) and pink backgrounds are conditioned on all circles (short attack range). Smaller sizes represent less shield and health. More examples shown in Appendix A.4.9.

enemies before initiating an attack. Conversely, agents with large attack ranges are advised to attack their enemies from a distance to ensure their own safety. Figure 2.5 shows that if we condition on all circles, the diffusion model thinks players will form a cluster and if condition on all squares, it will predict the players to attack each other from a distance, aligning well with the optimal policy. This demonstrates conditioned diffusion models’ ability to *implicitly decompose states to learn underlying compositions* and *capture multimodality of different unit behavior* in the training data. This highlights its ability to accurately predict the world model for OOC scenarios.

**Takeaway 3:** Conditioned diffusion models show significant promise by effectively decomposing and capturing modes of individual base elements and composing them according to new conditioning, which helps them to accurately predict the world dynamics and generalize to OOC scenarios.

## 2.7 Ablations

In this section, we ablate over our design choices to (1) show the necessity of using the inductive bias of state latent vector as conditioning, (2) different model architectures to incorporate conditioning information

### 2.7.1 Necessity of Combinatorial Inductive Bias

We compare trajectories generated by the conditioned and unconditioned diffusion models in this section to demonstrate the importance of using combinatorial latent information as conditioning. In Maze2D [20], we formulate the navigation problem as a one-step generation process where the diffusion model learns how to generate an entire valid trajectory without rolling out current action and replan. Since there is only one planning step in this process, generating a valid trajectory can be seen as denoising a line that connects the start and goal position, instead of generating action based on the current location. The inductive bias we use is every training trajectory will pass through three waypoints that equally slice the trajectory. In this case, the set of all waypoints forms the base element set and their combination is the latent vector that determines the shape of a generated maze trajectory. During training, we extract three points that equally slice the trajectory and use them as conditioning. During test time, we specify a new combination of three waypoints we want the generated trajectory to pass.

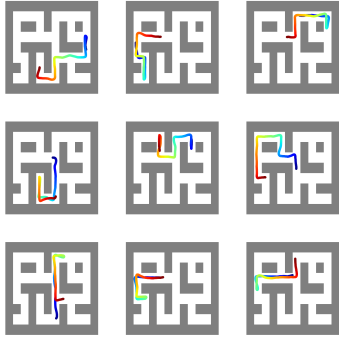


Figure 2.6: train traj

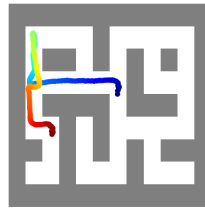


Figure 2.7: ID (Uncond)

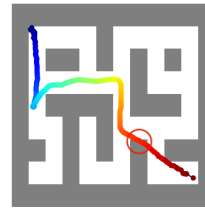


Figure 2.8: OOC (Uncond)

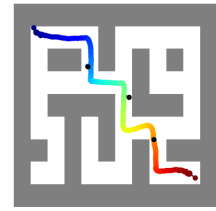


Figure 2.9: cfg=1.3 (Cond)

Figure 2.10: Trajectories generated in Maze2D for large maze. (a) Samples from the training set. (b) Trajectories directly generated by the unconditioned diffusion model given in distribution start and end positions. (c) Trajectories directly generated by the unconditioned diffusion model on unseen start and end positions. (d) Trajectories directly generated by a conditioned diffusion model using 3 waypoints (black dots) as conditioning with classifier-free guidance (cfg) weight 1.3. For results in medium maze please refer to Appendix A.4.6.

We see that the unconditioned diffusion model successfully generated a trajectory if the start and end positions are in the training set (Figure 2.7) but failed for unseen start and end points (Figure 2.8). However, if we use a conditioned diffusion model

and condition on an unseen combination of the three waypoints, it can generate unseen trajectories that still satisfy constraints supported by the training dataset (Figure 2.9). This highlights the importance of using conditioning.

### 2.7.2 Model Architecture: Attention vs Concatenation

We also ablate over different model architectures: (1) concatenating the latent vector  $\mathbf{z}$  with diffusion’s time embedding, (2) performing cross attention between  $\mathbf{z}$  and output of each Unet residual block (Architecture in Figure A.2). Concatenation and attention-based networks have roughly the same number of parameters as shown in Table A.17. Figure 2.11 shows our result: in general conditioned diffusion models outperform unconditioned ones and attention outperforms concatenation in 3 out of 4 cases.

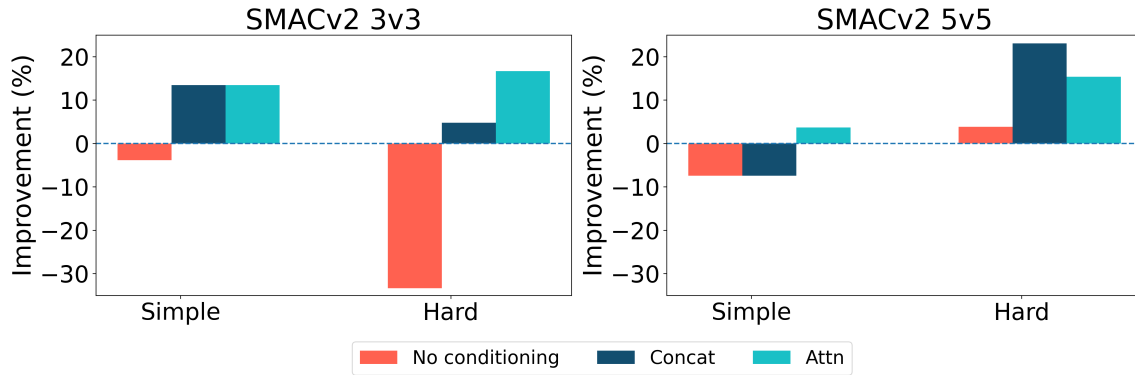


Figure 2.11: Improvement percentage over MAPPO for different types of conditioning in SMACv2.

**Ablation Takeaway:** Conditioned diffusion models, provided with information about the new composition of state, can generate better trajectories than unconditioned diffusion models. Also, cross-attention with the condition vector outperforms simply concatenating it with the time-embedding in most cases.

## 2.8 Conclusion

Despite the success of traditional RL models in decision-making tasks, they still struggle to generalize to unseen state inputs. Most existing work focuses on RL generalization under the assumption that generalization to a different probability

density function with the same support or allows unseen base elements but also introduces other potentially unrealistic assumptions. However, we take it further and study the problem of generalization to out-of-support states, out of combination in particular, hoping the model can exploit the compositional nature of our world. We showed how this task is challenging for many traditional RL algorithms that rely on accurate value prediction and also how conditioned diffusion models can generalize to unsupported samples. We compare the models in different environments with detailed ablation and analysis, demonstrating how each of these classic environments can be formulated as a state combinatorial problem.

However, one limitation of our setup is this combination of base elements formulation is less effective if environment complexity does not stem from exponentially many element combinations. Also, the distinction between different objects can be blurry (e.g. would a motorbike be a bike). Another constraint is efficiency, as planning with diffusion models in stochastic environments requires denoising a trajectory at each planning step, which can be computationally intensive.

## Chapter 3

# Accelerating Diffusion Models in Offline RL via Reward-Aware Consistency Trajectory Distillation

### 3.1 Motivation

Recent advances in diffusion models have demonstrated their remarkable capabilities across various domains [9, 34, 39, 49, 73], including decision-making tasks in reinforcement learning (RL). These models excel particularly in capturing multi-modal behavior patterns [3, 9, 34] and achieving strong out-of-distribution generalization [6, 16], making them powerful tools for complex decision-making scenarios. However, their practical deployment faces a significant challenge: the computational overhead of the iterative sampling procedures, which requires numerous denoising steps to generate high-quality outputs.

To address this limitation, various diffusion acceleration techniques have been proposed, including ordinary or stochastic differential equations (ODE or SDE) solvers with flexible step sizes [39, 51, 73], sampling step distillation [41, 76] and improved noise schedules and parametrizations [68, 74]. In particular, consistency distillation [76] has emerged as one of the most promising solutions for image generation, in which a many-step diffusion model serves as a teacher to train a student consistency

model that achieves comparable performance while enabling faster sampling through a single-step or few-step generation process.

This breakthrough has sparked considerable interest in applying consistency distillation to decision-making tasks. However, current applications either adopt a behavior cloning approach [52, 64, 81] or integrate few-step diffusion based samplers in actor-critic frameworks [8, 15, 47]. While promising, these approaches face inherent challenges: behavior cloning performs well only with expert demonstrations but struggles with suboptimal data (e.g., median-quality replay buffers), while the actor-critic paradigm introduces concurrent training of multiple networks and sensitive hyperparameters, exacerbating training complexity, instability, and computational overhead.

This raises an important question: can we develop a more effective approach to consistency distillation specifically tailored for offline RL? We address this challenge by introducing a novel method that directly incorporates reward optimization into the consistency distillation process. Our approach begins with a pre-trained unconditional diffusion policy and augments the standard consistency trajectory distillation [41] with an explicit reward objective. The vanilla consistency trajectory distillation helps the student model cover the diverse behavior patterns learned by the teacher. The additional objective encourages the student consistency model to sample actions that yield higher rewards, effectively steering the model toward selecting optimal trajectories from the multi-modal distributions captured by the teacher diffusion model.

Another key advantage of our method lies in its training simplicity. The reward model can be trained independently from the teacher diffusion model and the distillation process, avoiding the complexity of concurrent multi-network training present in actor-critic methods. Our method also eliminates the need for training noise-aware reward models, unlike existing guided diffusion sampling approaches [34]. By integrating reward optimization directly into the distillation process, our method achieves superior performance, enables efficient single-step generation, and maintains straightforward training procedures.

We demonstrate the performance and sampling efficiency of our RACTD on the sub-optimal heterogeneous D4RL Gym-MuJoCo benchmark and challenging long-horizon planning task Maze2d [20]. Our method demonstrates both superior performance and



and  $\gamma \in [0, 1]$  is a discount factor. In offline RL, we further assume that the agent can no longer interact with the environment and is restricted to learning from a size  $M$  static dataset  $\mathcal{D} = \{\tau_i\}_{i=1}^M$ , where  $\tau = (\mathbf{s}_0, \mathbf{a}_0, r_0, \mathbf{s}_1, \mathbf{a}_1, r_1, \dots, \mathbf{s}_H, \mathbf{a}_H, r_H)$  represents a rollout of episode horizon  $H$  collected by following a behavior policy  $\pi_\beta$ .

Mathematically, we want to find a policy  $\pi^*$  that

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[ \sum_{n=0}^H \gamma^n R(\mathbf{s}_n, \mathbf{a}_n) \right] \quad (3.1)$$

subject to the constraint that all policy evaluation and improvement must rely on  $\mathcal{D}$  alone.

### 3.2.2 Diffusion Models

Diffusion models generate data by learning to reverse a gradual noise corruption process applied to training examples. Given a clean data sample  $\mathbf{x}_0$ , we define  $\mathbf{x}_t$  for  $t \in [0, T]$  as increasingly noisy versions of  $\mathbf{x}_0$ . The forward (or noising) process is commonly formulated as an Itô stochastic differential equation (SDE):

$$d\mathbf{x} = f(\mathbf{x}, t)dt + g(t)dw \quad (3.2)$$

where  $w$  is a standard Wiener process. As  $t$  approaches the final timestep  $T$ , the distribution of  $\mathbf{x}_T$  converges to a known prior distribution, typically Gaussian. At inference time, the model reverses this corruption process by following the corresponding reverse-time SDE, which depends on the score function  $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ . In practice, this score function is approximated by a denoiser network  $D_\phi$ , enabling iterative denoising from  $\mathbf{x}_T$  back to  $\mathbf{x}_0$ . An alternative, deterministic interpretation of the reverse process is given by the probability flow ODE (PFODE):

$$d\mathbf{x} = \left[ f(\mathbf{x}, t) - \frac{1}{2}g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) \right] dt \quad (3.3)$$

which preserves the same marginal distribution  $p_t(\mathbf{x})$  as the reverse SDE at each timestep  $t$ . This ODE formulation often enables more efficient sampling through larger or adaptive step sizes without significantly compromising the sample quality.

EDM [39] refine both the forward and reverse processes through improved noise parameterization and training objectives. Concretely, they reparametrize the denoising score matching (DSM) loss so that the denoiser network learns to predict a scaled version of the clean data:

$$\mathcal{L}_{\text{EDM}} = \mathbb{E}_{t, \mathbf{x}_0, \mathbf{x}_t | \mathbf{x}_0} [d(\mathbf{x}_0, D_\phi(\mathbf{x}_t, t))] \quad (3.4)$$

where  $d$  is a distance metric in the clean data space. In this paper, we train an EDM model as the teacher using the pseudo huber loss as  $d$  following Prasad et al. [64].

At inference time, EDM solves the associated PFODE with a 2nd-order Heun solver.

### 3.2.3 Consistency Trajectory Distillation

The iterative nature of the diffusion sampling process introduces significant computational overhead. Among various acceleration techniques proposed, consistency distillation [76] has emerged as a particularly effective approach. The core idea is to train a student model that can emulate the many-step denoising process of a teacher diffusion model in a single step.

Building upon this framework, Kim et al. [41] introduced Consistency Trajectory Models (CTM). Instead of learning only the end-to-end mapping from noise to clean samples, CTM learns to predict across arbitrary time intervals in the diffusion process. Specifically, given three arbitrary timesteps  $0 \leq k < u < t \leq T$ , CTM aims to align two different paths to predict  $\mathbf{x}_k$ : (1) direct prediction from time  $t$  to  $k$  using the student model, and (2) a two-stage prediction that first uses a numerical solver (e.g., Heun) with the teacher model to predict from time  $t$  to  $u$ , and then uses the student model to predict from time  $u$  to  $k$ .

Since the distance metric  $d$  is defined on the clean data space and may not be well-defined in the noisy data space, in practice we further map all predictions to time 0 using the student model. Formally, denote the student model as  $G_\theta$ , the CTM loss is defined as:

$$\mathcal{L}_{\text{CTM}} = \mathbb{E} \left[ d \left( G_{sg(\theta)}(\hat{\mathbf{x}}_k^{(t)}, k, 0), G_{sg(\theta)}(\mathbf{x}_k^{(t,u)}, k, 0) \right) \right] \quad (3.5)$$

where  $G_\theta(\mathbf{x}_t, t, u)$  represents the student prediction from time  $t$  to  $u$  given noisy

### 3. Accelerating Diffusion Models in Offline RL via Reward-Aware Consistency Trajectory Distillation

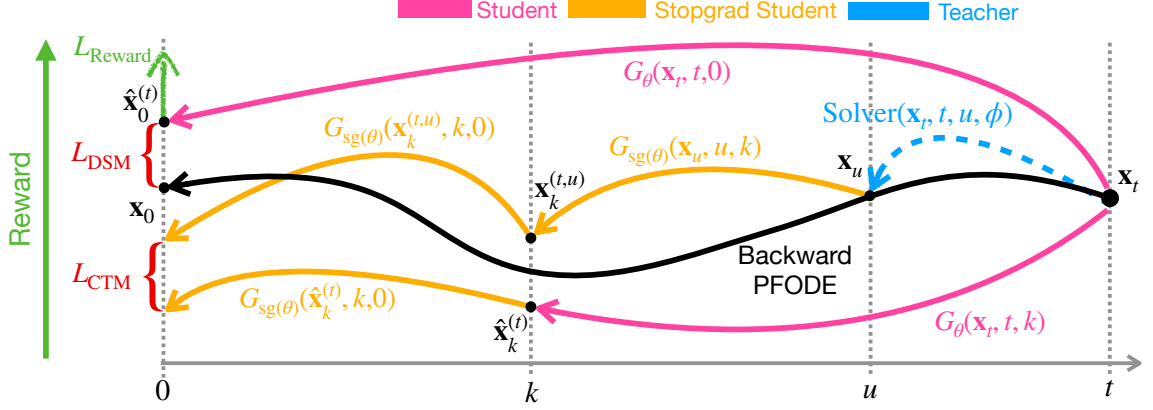


Figure 3.2: Visualization of CTM loss, DSM loss and reward loss.

sample  $\mathbf{x}_t$  at time  $t$ ,  $\text{sg}(\theta)$  represents stop-gradient student parameters and

$$\hat{\mathbf{x}}_k^{(t)} = G_\theta(\mathbf{x}_t, t, k), \quad \mathbf{x}_k^{(t,u)} = G_{\text{sg}(\theta)}(\text{Solver}(\mathbf{x}_t, t, u; \phi), u, k) \quad (3.6)$$

Here  $\text{Solver}(\mathbf{x}_t, t, u; \phi)$  is the numerical solver result from time  $t$  to  $u$  using the teacher model  $D_\phi$  given noisy sample  $\mathbf{x}_t$  at time  $t$ .

In addition to the CTM loss, consistency trajectory distillation also incorporates the DSM loss to enforce its samples to be close to the training data. The DSM loss for the student model is the same as the one for EDM in Equation 3.4:

$$\mathcal{L}_{\text{DSM}} = \mathbb{E}_{t, \mathbf{x}_0, \mathbf{x}_t | \mathbf{x}_0} [d(\mathbf{x}_0, G_\theta(\mathbf{x}_t, t, 0))] \quad (3.7)$$

In Figure 3.2 we provide a visualization of these objectives on a PFODE trajectory. After the distillation, the student model can perform “anytime-to-anytime” jumps along the PFODE trajectory. One-step sampling can then be achieved by calculating  $\hat{\mathbf{x}}_0^{(T)} = G_\theta(\mathbf{x}_T, T, 0)$ .

### 3.3 Method

#### 3.3.1 Motivation and Intuition

Diffusion models and their consistency-based counterparts have demonstrated promising results in decision-making tasks, particularly in capturing multimodal behavior patterns [3, 9, 34]. Common recipes for using these models in decision-making generally fall into one of the three paradigms: (1) training a diffusion or consistency model via behavior cloning and deploying it directly as a policy [3, 9]; (2) integrating the model into actor-critic frameworks [15, 26, 80]; or (3) using the model as a planner through guided diffusion sampling [34].

While behavior cloning pipelines perform well when trained on expert demonstrations, they often struggle with suboptimal datasets (e.g., medium-replay buffers) collected from diverse behavior policies. Such datasets typically exhibit complex multimodal behavior patterns, where only some modes lead to high rewards. Although one could potentially use rejection sampling to filter out low-reward training data, this approach becomes prohibitively sample inefficient, particularly as the quality of the training data deteriorates. On the other hand, to generate high reward actions, actor-critic approaches require concurrent training of multiple neural networks with sensitive hyperparameters. Finally, guided diffusion sampling necessitates training noise-aware reward models and multi-step sampling, which could be detrimental for time-sensitive decision-making tasks like self-driving.

So how can we better leverage potentially suboptimal datasets to design a diffusion-based single-step sampling model with simple training procedure? Our key idea is to utilize the multimodal information captured by the teacher model and encourage the student model to favor the high reward modes. We achieve this by incorporating a reward objective directly in the consistency distillation process. Since our student model can achieve single-step denoising, we can incorporate a reward model trained in the clean sample space and avoid the multi-step reward optimization for diffusion models.

### 3.3.2 Modeling Action Sequences

Before introducing our reward-aware consistency trajectory distillation, we would like to first clearly establish the modeling formulation in our method. When applying diffusion models to RL, several modeling choices are available: modeling actions (as a policy), modeling rollouts (as a planner), or modeling state transitions (as a world model). While more comprehensive modeling approaches can offer advantages, particularly in long-horizon tasks, they also introduce additional complexity and computational challenges.

Following Chi et al. [9], we adopt a balanced approach: modeling a fixed-length sequence of future actions conditioned on a fixed-length sequence of observed states. This formulation ensures consecutive actions form coherent sequences, and reduces the chances of generating idle actions.

Formally, let  $\vec{s}_n = \{s_{n-h}, s_{n-h+1}, \dots, s_n\}$  denote a length- $h$  sequence of past states at rollout time  $n$ , and  $\vec{a}_n = \{a_n, a_{n+1}, \dots, a_{n+c}\}$  represent a length- $c$  sequence of future actions. Both the teacher and the student learn to model the conditional distribution  $p(\vec{a}_n | \vec{s}_n)$ . In the context of diffusion notations,  $\mathbf{x} = \vec{a}_n | \vec{s}_n$ . We can easily extend this framework for goal-conditioned RL, where  $\vec{s}_n = \{s_n, s_T\}$  are the current and goal state that is used for conditioning.

During execution, we can either execute only the first predicted action  $a_n$  in the environment before replanning at the next step, or follow the entire predicted sequence of actions at once.

### 3.3.3 Reward-Aware Consistency Trajectory Distillation

Having established the formulation, we now present our approach to integrating reward optimization into the consistency trajectory distillation process.

Let  $R_\psi$  be a pre-trained differentiable return-to-go network (i.e. reward model) that takes the state  $s_n$  and action  $a_n$  at rollout time  $n$  as inputs and predicts the future discounted cumulative reward  $\hat{r}_n = \sum_{j=0}^{H-n} \gamma^j r_{n+j}$ .

When the student model generates a prediction  $\vec{a}_n | \vec{s}_n = \hat{\mathbf{x}}_0^{(T)} = G_\theta(\mathbf{x}_T, T, 0)$ , we extract the action at time  $n$ , denoted as  $\hat{a}_n$ , from the predicted sequence and pass it along with  $s_n$  to the frozen reward model  $R_\psi$  to estimate  $\hat{r}_n$ . The goal of our reward-aware training is to maximize the estimated discounted cumulative reward.

Mathematically, the reward objective is defined as

$$\mathcal{L}_{\text{Reward}} = -R_{\psi}(\vec{s}_n, \hat{\mathbf{a}}_n) \quad (3.8)$$

The final loss for reward-aware consistency trajectory distillation (RACTD) combines all three terms:

$$\mathcal{L} = \alpha \mathcal{L}_{\text{CTM}} + \beta \mathcal{L}_{\text{DSM}} + \sigma \mathcal{L}_{\text{Reward}} \quad (3.9)$$

where  $\alpha$ ,  $\beta$ , and  $\sigma$  are hyperparameters to balance different loss terms.

### 3.3.4 Decoupled Training

A key advantage of our method of combining reinforcement learning, diffusion models, and consistency distillation is the ability to support fully decoupled training of all components.

Traditional actor-critic frameworks, which are commonly used to incorporate diffusion models into reinforcement learning, require simultaneous training of multiple neural networks. This concurrent optimization presents considerable challenges, often demanding extensive hyperparameter tuning and careful balancing of different learning objectives.

Guided diffusion sampling, as proposed in Janner et al. [34], offers an alternative approach by taking inspiration from classifier guided diffusion [14, 75]. However, these classifiers (i.e. reward models) require noise-aware training that cannot be separated from the diffusion model. Also, predicting the correct reward from highly corrupted input could be very challenging, which can lead to inaccurate guidance that accumulates during its multi-step sampling process.

Our method, in contrast, fully leverages the advantages of single-step denoising models by operating entirely in the noise-free state-action space. This design choice enables the reward model to provide stable and effective signals without requiring noise-aware training. Importantly, the reward model can be pre-trained completely decoupled from the teacher model and distillation process. This separation not only simplifies the training process but also allows for flexible integration of different reward models using the same teacher model.

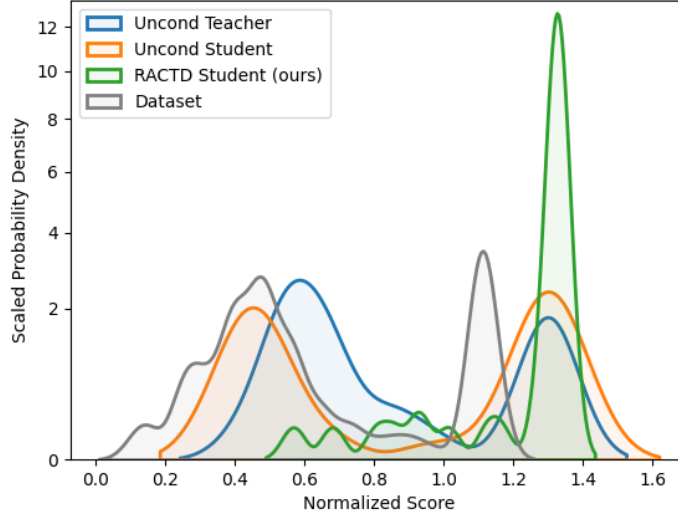


Figure 3.3: The reward distribution of the D4RL hopper-medium-expert dataset and 100 rollouts from an unconditioned teacher, an unconditioned student, and RACTD.

### 3.3.5 Reward Objective as Mode Selection

In offline RL, models often have to learn from datasets containing behaviors of varying quality. While diffusion models excel at capturing these diverse behavioral modes, they inherently lack the ability to differentiate between actions that lead to high versus low rewards. Our RACTD addresses this limitation by transforming the reward-agnostic teacher diffusion sampling distribution into one that preferentially samples from high-reward modes. We empirically verify this through a comparative analysis using the D4RL hopper-medium-expert dataset [20], which contains an equal mixture of expert demonstrations and suboptimal rollouts from a partially trained policy.

Figure 3.3 illustrates the reward distributions of rollouts sampled from three models: the unconditioned teacher, unconditioned student, and RACTD. The dataset (grey) exhibits two distinct modes corresponding to medium-quality and expert rollouts. The unconditioned teacher model (blue) accurately captures this bimodal distribution, and the unconditioned student model (orange) faithfully replicates it. In contrast, our RACTD (green) concentrates its samples on the higher-reward mode, demonstrating that our reward guidance effectively identifies and selects optimal behaviors from the teacher’s multi-modal distribution. We also include the discussion

between sample diversity and mode selection in Appendix B.4.

## 3.4 Experiment

In this section, we conduct experiments to demonstrate: (1) the effectiveness of our RACTD, especially when the teacher model is trained with suboptimal data, (2) the capability of accurately capturing the complex and diverse high dimensional behavior patterns with only one denoising step, and (3) the speed-up achieved over the teacher model and existing policy-based diffusion models.

### 3.4.1 Offline RL

**Baselines** We compare our approach against a comprehensive set of baselines, including vanilla behavior cloning (BC) and Consistency Policy (Consistency BC) [15]; model-free RL algorithms CQL [43] and IQL [42]; model-based algorithms Trajectory Transformer (TT) [33], MOPO [85], MOReL [40], MBOP [4]; autoregressive model Decision Transformer (DT) [7]; diffusion-based planner Diffuser [34]; and diffusion-based actor-critic methods Diffusion QL [80] and Consistency AC [15].

**Setup** We first evaluate our method and the baselines on D4RL Gym-MuJoCo [20], which is a popular offline RL benchmark that contains three environments (hopper, walker2d, halfcheetah) of mixtures of varying quality data (medium-replay, medium, medium-expert).

Following the setups in the prior works, we employ online model selection for evaluation by using the best-performing checkpoint observed during training. Results for non-diffusion-based models and Diffuser are taken from Janner et al. [34], and results for Diffusion QL and Consistency AC/BC are sourced from their respective papers. The results for RACTD are reported as the mean and standard error over 100 planning seeds. We use  $h = 1, c = 16$  and closed-loop plannign in all experiments.

**Results** As shown in Table 3.1, RACTD achieves the best or second best performance in almost all task, and outperforms the best baseline in overall average score by a substantial margin. It consistently outperforms the diffusion-based planner Diffuser

Table 3.1: (Offline RL) Performance of RACTD and a variety of baselines on the D4RL Gym-MujoCo benchmark. The best score is emphasized in bold and the second-best is underlined.

| Method         | Medium Expert         |                        |                        | Medium                |                        |                       | Medium Replay         |                        |                       | Avg         | NFE      |
|----------------|-----------------------|------------------------|------------------------|-----------------------|------------------------|-----------------------|-----------------------|------------------------|-----------------------|-------------|----------|
|                | HalfCheetah           | Hopper                 | Walker2d               | HalfCheetah           | Hopper                 | Walker2d              | HalfCheetah           | Hopper                 | Walker2d              |             |          |
| BC             | 55.2                  | 52.5                   | 107.5                  | 42.6                  | 52.9                   | 75.3                  | 36.6                  | 18.1                   | 26.0                  | 51.9        | -        |
| CQL            | 91.6                  | 105.4                  | 108.8                  | 44.0                  | 58.5                   | 72.5                  | 45.5                  | 95.0                   | 77.2                  | 77.6        | -        |
| IQL            | 86.7                  | 91.5                   | 109.6                  | 47.4                  | 66.3                   | 78.3                  | 44.2                  | 94.7                   | 73.9                  | 77.0        | -        |
| DT             | 86.8                  | 107.6                  | 108.1                  | 42.6                  | 67.6                   | 74.0                  | 36.6                  | 82.7                   | 66.6                  | 74.7        | -        |
| TT             | 95.0                  | 110.0                  | 101.9                  | 46.9                  | 61.1                   | 79.0                  | 41.9                  | 91.5                   | 82.6                  | 78.9        | -        |
| MOPO           | 63.3                  | 23.7                   | 44.6                   | 42.3                  | 28.0                   | 17.8                  | 53.1                  | 67.5                   | 39.0                  | 42.1        | -        |
| MOReL          | 53.3                  | 108.7                  | 95.6                   | 42.1                  | 95.4                   | 77.8                  | 40.2                  | 93.6                   | 49.8                  | 72.9        | -        |
| MBOP           | <b>105.9</b>          | 55.1                   | 70.2                   | 44.6                  | 48.8                   | 41.0                  | 42.3                  | 12.4                   | 9.7                   | 47.8        | -        |
| Diffusion QL   | <u>97.2</u> $\pm 0.4$ | <u>112.3</u> $\pm 0.8$ | 111.2 $\pm 0.9$        | 51.5 $\pm 0.3$        | 96.6 $\pm 3.4$         | <u>87.3</u> $\pm 0.5$ | 48.3 $\pm 0.2$        | <u>102.0</u> $\pm 0.4$ | <b>98.0</b> $\pm 0.5$ | 89.3        | 5        |
| Consistency AC | 89.2 $\pm 3.3$        | 106.0 $\pm 1.3$        | <u>111.6</u> $\pm 0.7$ | <b>71.9</b> $\pm 0.8$ | <u>99.7</u> $\pm 2.3$  | 84.1 $\pm 0.3$        | <b>62.7</b> $\pm 0.6$ | 100.4 $\pm 0.6$        | 83.0 $\pm 1.5$        | <u>89.8</u> | 2        |
| Consistency BC | 39.6 $\pm 3.4$        | 96.8 $\pm 4.6$         | <u>111.6</u> $\pm 0.7$ | 46.2 $\pm 0.4$        | 78.3 $\pm 2.6$         | 84.1 $\pm 0.3$        | 45.4 $\pm 0.7$        | 100.4 $\pm 0.6$        | 80.8 $\pm 3.4$        | 75.9        | 2        |
| Diffuser       | 88.9 $\pm 0.3$        | 103.3 $\pm 1.3$        | 106.9 $\pm 0.2$        | 42.8 $\pm 0.3$        | 74.3 $\pm 1.4$         | 79.6 $\pm 0.55$       | 37.7 $\pm 0.5$        | 93.6 $\pm 0.4$         | 70.6 $\pm 1.6$        | 77.5        | 20       |
| RACTD(Ours)    | 95.9 $\pm 1.5$        | <b>129.0</b> $\pm 1.3$ | <b>119.6</b> $\pm 1.9$ | <u>59.3</u> $\pm 0.2$ | <b>121.0</b> $\pm 0.5$ | <b>98.9</b> $\pm 1.6$ | <u>57.9</u> $\pm 1.0$ | <b>104.9</b> $\pm 2.1$ | <u>91.5</u> $\pm 2.6$ | <b>97.6</b> | <b>1</b> |

in all environments, and exceeds the performance of consistency-based actor-critic baseline Consistency AC in all hopper and walker2d tasks.

### 3.4.2 Long Horizon Planning

Next, we showcase the effectiveness of RACTD in complex high-dimensional long-horizon planning and its flexibility to adapt to goal-conditioned tasks. Previously, Diffuser has shown great potential in open-loop long-horizon planning, but requires a significantly larger number of denoising steps compared to closed-loop planning like MuJoCo. We demonstrate that our model can achieve superior performance with a single-step denoising process under the same problem formulation.

**Setup** We test this ability on D4RL Maze2d [20], which is a sparse reward long-horizon planning task where an agent may take hundreds of steps to reach the goal in static environments. Following the setup in Janner et al. [34], we use a planning horizon 128, 256, 384 for U-Maze, Medium and Large respectively. We perform open-loop planning by generating the entire state sequence followed by a reverse dynamics model to infer all the actions from the predicted state sequence. The reward model returns 1 if the current state reaches the goal and 0 otherwise. The baseline results are reported from Janner et al. [34] and RACTD results are reported as the mean and standard error of 100 planning seeds.

Table 3.2: (Long-horizon planning) The performance of RACTD, Diffuser, and prior model-free algorithms in the Maze2D environment. The best score is in bold and the second-best is underlined.

| Method       | U-Maze                 |          |              | Medium                 |          |              | Large                  |          |              | Avg Score    |
|--------------|------------------------|----------|--------------|------------------------|----------|--------------|------------------------|----------|--------------|--------------|
|              | Score                  | NFE      | Time (s)     | Score                  | NFE      | Time (s)     | Score                  | NFE      | Time (s)     |              |
| MPPI         | 33.2                   | -        | -            | 10.2                   | -        | -            | 5.1                    | -        | -            | 16.2         |
| CQL          | 5.7                    | -        | -            | 5                      | -        | -            | 12.5                   | -        | -            | 7.7          |
| IQL          | 47.4                   | -        | -            | 34.9                   | -        | -            | 58.6                   | -        | -            | 47.0         |
| Diffuser     | 113.9 $\pm 3.1$        | 64       | 1.664        | <u>121.5</u> $\pm 2.7$ | 256      | 4.312        | 123.0 $\pm 6.4$        | 256      | 5.568        | 119.5        |
| CTD          | <u>123.4</u> $\pm 1.0$ | <b>1</b> | <b>0.029</b> | 119.8 $\pm 4.1$        | <b>1</b> | <b>0.047</b> | <u>127.1</u> $\pm 6.4$ | <b>1</b> | <b>0.049</b> | <u>123.4</u> |
| RACTD (Ours) | <b>125.7</b> $\pm 0.6$ | <b>1</b> | <b>0.029</b> | <b>130.8</b> $\pm 1.8$ | <b>1</b> | <b>0.047</b> | <b>143.8</b> $\pm 0.0$ | <b>1</b> | <b>0.049</b> | <b>133.4</b> |

**Results** As shown in Table 3.2, both Diffuser and RACTD outperform previous model-free RL algorithms CQL and IQL, and MPPI which uses ground-truth dynamics. Our approach surpasses the diffusion baseline in almost all settings, highlighting its ability to effectively capture complex behavioral patterns and high-dimensional information in the training dataset. Notably, the planning dimension for this task (384 for the Large Maze) is substantially higher than that of MuJoCo tasks (16). As a result, Diffuser requires significantly more denoising steps (256 for the Large Maze) compared to MuJoCo (20 steps). On the contrary, despite the increased task complexity, RACTD still only requires a single denoising step to achieve 11.6 $\times$  performance boost.

### 3.4.3 Inference Time Comparison

Beyond performance improvements, a key contribution of our work is significantly accelerating diffusion-based models for decision-making tasks. The primary computational bottleneck in diffusion models arises from the multiple function evaluations (NFEs) required by the iterative denoising process. By reducing the number of denoising steps to a single NFE, our approach achieves a speed-up roughly proportional to the number of denoising originally required.

**Setup** To evaluate sampling efficiency, we compare RACTD with different samplers, including DDPM [28], DDIM [73], and EDM [39] using the same network architecture as our teacher model. Additionally, we report the efficiency of Diffuser. Note that since

Table 3.3: Wall clock time and NFEs per action for different samplers and Diffuser on MuJoCo hopper-medium-replay.

| Method                | Time (s)     | NFE      | Score        |
|-----------------------|--------------|----------|--------------|
| Diffuser              | 0.644        | 20       | 93.6         |
| DDPM                  | 0.236        | 15       | 24.2         |
| DDIM                  | 0.208        | 15       | 60.6         |
| EDM (Teacher)         | 2.134        | 80       | <b>114.2</b> |
| RACTD (Ours, Student) | <b>0.015</b> | <b>1</b> | 104.9        |

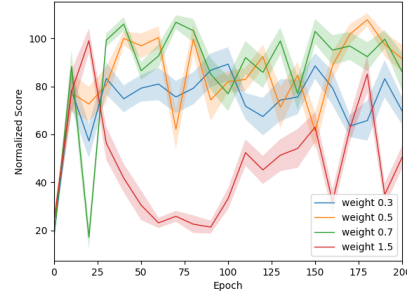


Figure 3.4: Ablation on reward objective weight.

Diffuser employs a different model architecture and generates future state-action pairs, its sampling time may also be influenced by these factors. Table 3.3 and Table B.6 present the wall clock sampling time and NFE for MuJoCo (hopper-medium-replay) and Maze2d. All experiments are conducted on one NVIDIA Tesla V100-SXM2-32GB.

**Results** In hopper-medium-replay, RACTD achieved  $20\times$  reduction in NFEs and a  $43\times$  speed-up compared to Diffuser. Additionally, our student model requires  $80\times$  fewer NFEs and samples  $142\times$  faster than the teacher model. In Maze2d, RACTD significantly accelerates computation compared to Diffuser, achieving approximately  $57\times$ ,  $92\times$ , and  $114\times$  speed-ups on Umaze, Medium and Large mazes, by reducing NFEs by a factor of 256 for the Medium and Large mazes. Furthermore, we observe that more denoising steps lead to better model performance. This highlights the advantage of distilling from a slow but high-performing teacher model, which enables better performance compared to training a diffusion model directly with fewer denoising steps.

## 3.5 Ablation Study

We ablate over (1) the impact of the reward objective in both student and teacher model training, (2) the effect of different reward loss weights on model performance and training stability, and (3) the impact of increasing denoising steps with our student model.

Table 3.4: We compare incorporating the reward model in different stages of training on MuJoCo hopper-medium-replay. Results are presented as the mean and standard error across 100 seeds.

| <b>Hopper<br/>Medium-Replay</b> | Unconditioned<br>teacher | Reward-Aware<br>teacher |
|---------------------------------|--------------------------|-------------------------|
| Unconditioned student           | 50.8 $\pm$ 0.3           | 96.2 $\pm$ 0.2          |
| Reward-Aware student            | <b>104.9</b> $\pm$ 2.1   | 96.0 $\pm$ 0.3          |

Table 3.5: Inference time, NFE, and score comparison for student model multi-step sampling on MuJoCo hopper-medium-replay.

| <b>NFE</b> | <b>Time(s)</b> | <b>Score</b>           |
|------------|----------------|------------------------|
| 1          | <b>0.0147</b>  | 104.9 $\pm$ 2.1        |
| 2          | 0.0241         | <b>109.8</b> $\pm$ 0.9 |
| 3          | 0.0377         | 108.7 $\pm$ 0.1        |
| 4          | 0.0517         | 107.9 $\pm$ 1.6        |

### 3.5.1 Impact of Reward Objective

To understand the unique advantages of incorporating reward objectives during distillation, we conduct a systematic comparison across four model configurations: the baseline combination of an unconditioned teacher and student, a reward-aware teacher paired with an unconditioned student, a fully reward-aware teacher-student pair, and our proposed RACTD, which combines an unconditioned teacher with a reward-aware student. The results for hopper-medium-replay and walker-medium is shown in Table 3.4 and Table B.5.

Our analysis reveals that while incorporating reward objectives at any stage yields substantial improvements, optimal performance is achieved through our RACTD framework, which combines an unconditioned teacher with reward-aware student distillation. Our method allows the teacher to capture a comprehensive range of behavioral patterns, while enabling the student to selectively distill the most effective strategies. Although incorporating reward objectives in the teacher model also enhances performance, this approach risks discarding suboptimal behaviors that may be valuable in novel testing scenarios, potentially limiting the model’s generalization capabilities.

### 3.5.2 Effect of Reward Objective Weights

The reward loss weight is a crucial hyperparameter in our pipeline that impacts both training stability and performance. We plot the reward curve achieved over 200 epochs of student training with reward loss weights ranging from [0.3, 0.5, 0.7, 1.5]

on MuJoCo hopper-medium-replay in Figure 3.4. The mean and standard error are reported across 20 rollouts from intermediate checkpoints.

With lower weights, increasing the weight leads to higher performance and relatively stable training. However, when the weight is too high (e.g. 1.5 in this plot), evaluation initially increases but fluctuates as training progresses. This occurs when the reward loss dominates DSM and CTM losses, resulting in unstable training.

### 3.5.3 Number of Sampling Steps

Since our student model is trained for anytime-to-anytime jumps, it naturally extends to multi-step denoising without additional training. Following the approach in Song et al. [76], given intermediate denoising timesteps  $0 < t_1 < t_2 < T$ , we first denoise from  $T$  to 0 as usual. We then add noise again to  $t_1$  and denoise it back to 0, and repeat this process for  $t_2$ . This iterative refinement can enhance generation quality. We evaluate the student using 2, 3, and 4 denoising steps as reported in Table 3.5. While multi-step sampling improves performance, we observe that gains do not scale linearly with the number of denoising steps.

## 3.6 Related Work

**Diffusion Models in Reinforcement Learning** Diffusion models have emerged as a powerful approach for decision-making tasks in reinforcement learning [3, 9, 26, 34, 80]. The integration of diffusion models into RL frameworks typically follows two main paradigms: actor-critic approaches, where diffusion models serve as policy or value networks [26, 80], and policy-based approaches, where diffusion models directly generate action trajectories [3, 9, 34]. While these methods have demonstrated impressive performance on standard RL benchmarks, their practical deployment is hindered by the slow sampling time inherent to vanilla diffusion policies based on DDPM [28]. This limitation poses particular challenges for real-world applications such as robotics, where speed is crucial.

**Accelerating Diffusion Model Sampling** Various approaches have been proposed to accelerate the sampling process in diffusion models. One prominent direction

leverages advanced ODE solvers to reduce the number of required denoising steps [39, 51, 73]. Another line of work explores knowledge distillation techniques [5, 41, 54, 68, 76], where student models learn to take larger steps along the ODE trajectory. Particularly, consistency models [76] leverage the PFODE to learn direct mappings from noise to sample space. Consistency trajectory models [41] enable one-step sampling by learning anytime-to-anytime jumps along the PFODE trajectory, achieving performance that surpasses teacher models.

**Consistency Models in Decision Making** Consistency models have emerged as a promising policy class for behavior cloning from expert demonstrations in robotics [52, 64, 81]. In RL, several works have enhanced actor-critic methods by replacing traditional diffusion-based value/policy networks with consistency models, showing faster inference and training speed [8, 15, 47]. These approaches directly incorporate consistency loss [76] into the value/policy network training, rather than distilling a separate student model. While Wang et al. [79] made some initial attempts to apply consistency distillation in policy-based RL through classifier-free guidance and reverse dynamics, their approach requires two NFEs and under-performs the state-of-the-art. In contrast, RACTD is a straightforward approach of using a separate reward model and incorporating reward objective during student distillation, achieving superior performance with only one NFE.

## 3.7 Conclusion

In this section, we address the challenge of diffusion policy acceleration by introducing reward-aware consistency trajectory distillation (RACTD), which predicts high-reward actions in a single denoising step. RACTD uses a pre-trained diffusion teacher model and a separately trained reward model, leveraging the teacher’s ability to capture multi-modal distributions while prioritizing higher-reward modes to generate high-quality samples from suboptimal training data. Its decoupled training approach avoids the complex concurrent optimization of multiple networks and enables the use of a standalone, noise-free reward model. RACTD outperforms previous state-of-the-art by 8.7% while accelerating its diffusion counterparts up to a factor of 142.

### *3. Accelerating Diffusion Models in Offline RL via Reward-Aware Consistency Trajectory Distillation*

# Chapter 4

## Conclusions

This thesis explored how diffusion models can serve as both generalizable and efficient policies for decision-making tasks. We first study the problem of generalization to out-of-support states, out of combination in particular, hoping the model can exploit the compositional nature of our world. We showed how this task is challenging for many traditional RL algorithms that rely on accurate value prediction and also how conditioned diffusion models can generalize to unsupported samples. We compare the models in different environments with detailed ablation and analysis, demonstrating how each of these classic environments can be formulated as a state-combinatorial problem.

To address the efficiency limitations of diffusion-based planning, we introduce reward-aware consistency trajectory distillation (RACTD), which predicts high-reward actions in a single denoising step. RACTD uses a pre-trained diffusion teacher model and a separately trained reward model, leveraging the teacher’s ability to capture multi-modal distributions while prioritizing higher-reward modes to generate high-quality samples from suboptimal training data. Its decoupled training approach avoids the complex concurrent optimization of multiple networks and enables the use of a standalone, noise-free reward model. RACTD outperforms previous state-of-the-art by 8.7% while accelerating its diffusion counterparts up to a factor of 142.

Overall, this thesis demonstrates that diffusion models, when properly adapted, offer a powerful framework for decision-making that combines generalization with efficiency, paving the way for scalable and robust learning in complex environments.

#### 4. *Conclusions*

# Appendix A

## State Combinatorial Generalization In Decision Making With Conditional Diffusion Models

### A.1 Related Work

**Meta RL** Meta RL is often seen as the problem of “learning to learn”, where agents are trained on several environments sampled from a task distribution during meta-training and tested on environments sampled from the same distribution during meta-testing [19, 84]. In the K-shot meta-RL setting, the model can interact with the testing environment K times during meta-testing time to update the model using reward [19, 48, 59, 67]. Our setting is different from Meta-RL as the training and testing environments are sampled from different distributions and conditioning is provided while restricting K to zero.

Attached below is a review of RL environments for generalization.

#### A.1.1 Generalization in RL

**Environments** Most RL environments that test model generalization can be grouped into different reward functions [19, 66, 67, 86] or transition functions [13, 56, 63, 86], goals or tasks [19, 84], states [11, 12, 22, 25, 35, 44, 57, 61]. For environments

with different state distributions, randomization [22] and procedural generation [12, 44, 61] are widely used to generate new states. Some vision-based environments [25, 35, 57] also use different rendering themes or layouts to generate unseen observations, more targeting sim2real problems. For robotics benchmarks like Metaworld [84] and RLbench [32], how much structure is shared between tasks like open a door and open a drawer is ambiguous [1]. Also, benchmarks like Franka Kitchen [23] focus on composing tasks at time horizons, requiring the model to concatenate trajectories corresponding to different subtasks. However, despite the large volume of generalization benchmarks, there is no benchmark designed for state combinatorial generalization to our best knowledge.

### **A.1.2 Combinatorial Generalization in Computer Vision**

**Computer Vision** The closest line of work to ours is combinatorial generalization for image generation where the model needs to learn new combinations of a discrete set of basic concepts like color and shapes and generalize to unseen combinations [31, 62, 70, 82]. This problem is often approached with disentangled representation learning [50, 70] with models like VAE but little evidence shows they can fully exhibit generalization ability [60, 70]. Okawa et al. [62] studied the capabilities of conditioned diffusion models on a synthetic shape generation task and showed that their composition ability emerges with enough training, first to closer concepts, then to farther ones.

## A.2 Proof of Corollary 5.1

**Corollary A.2.1** (Corollary 5.1). *Suppose the states lie along a linear manifold  $\mathcal{M}$  in the state space  $\mathbf{S}$  and the latent space  $\mathbf{Z}$  is well constructed so that  $\mathbf{Z}$  is (affine) isomorphic to  $\mathcal{M}$ . Let  $\mathbf{s}$  be a state in the training set with corresponding latent vector  $\mathbf{z}$  and  $\mathbf{s}'$  be an OOC state with corresponding latent vector  $\mathbf{z}'$ . Then a diffusion model  $p_\theta$  that is well trained on  $P_{\text{train}}$  can sample  $\mathbf{s}'$  with non-zero probability.*

*Proof.* We prove Corollary 2.4.1 by construction. Suppose  $\mathcal{M}$  is a  $d$ -dimensional linear manifold and  $\mathbf{S} \subset \mathbb{R}^k$ , then we note that both  $\mathcal{M}$  and  $\mathbf{Z}$  are affine isomorphic to  $\mathbb{R}^d$ . Therefore, with necessary shifting, we can have  $\mathbf{z}' = \mathbf{z} + \mathbf{y}$  where  $\mathbf{y} \in \mathbf{Z}$ . Let  $\mathbf{v}$  be the corresponding vector of  $\mathbf{y}$  in  $\mathcal{M}$ .

Now let's perform DDIM inversion on a training sample  $\mathbf{s}$  to obtain the SDE trajectory  $\{\mathbf{s}_t\}$ . Let  $\gamma_t$  be the angle between  $\epsilon_\theta(\mathbf{s}_t, t)$  and  $\mathcal{M}$ , there exist a set of vectors  $\mathbf{v}_t$  such that  $\mathbf{v} = \sum_t \sin(\gamma_t) \sigma_t \mathbf{v}_t$  and  $\mathbf{v}_t$  perpendicular to  $\epsilon_\theta(\mathbf{s}_t, t)$ . Then by construction and with necessary shifting, the trajectory  $\mathbf{s}'_t = \mathbf{s}_t + \mathbf{v}_t$  is a valid diffusion denoising trajectory (with  $\mathbf{v}_t$  acting as the “random” vector  $\epsilon_t$  sampled at each time step). This trajectory will yield  $\mathbf{s}'$  as the final state with non-zero probability because each intermediate Gaussian distribution  $p_\theta(\mathbf{s}_{t-1}|\mathbf{s}_t) = \mathcal{N}(\mathbf{s}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{s}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{s}_t, t))$  is defined on the entire ambient space. By [78] we know that  $\gamma_t \rightarrow \pi/2$  as  $t \rightarrow 0$ , therefore  $\mathbf{v}$  can be a non-zero vector. Hence, there exists a  $\mathbf{v} \in \mathcal{M}$  such that the sampling probability of  $\mathbf{s}' = \mathbf{s} + \mathbf{v}$  from diffusion model  $p_\theta$  is non-zero.  $\square$

While we have proven non-zero probability above, one can easily spot that, the probability can become extremely close to zero if the OOC sample is very far away from all training examples due to the intermediate Gaussian distributions. This corresponds to generalizing to the out-of-distribution samples with unseen base elements (the gray area in Figure 1). One can mitigate this problem by increasing the coverage of the support of the training space, which is also a common method in traditional RL to mitigate the problem of generalization to unseen base elements. Applying other post-training sampling techniques like repainting [53] can also allow extra Langevin steps when  $t$  is large.

### A.3 Additional Visualization of Value Function of PPO

We include here the value prediction of PPO for in-distribution and OOC states to demonstrate that OOC states are also a problem for online methods.

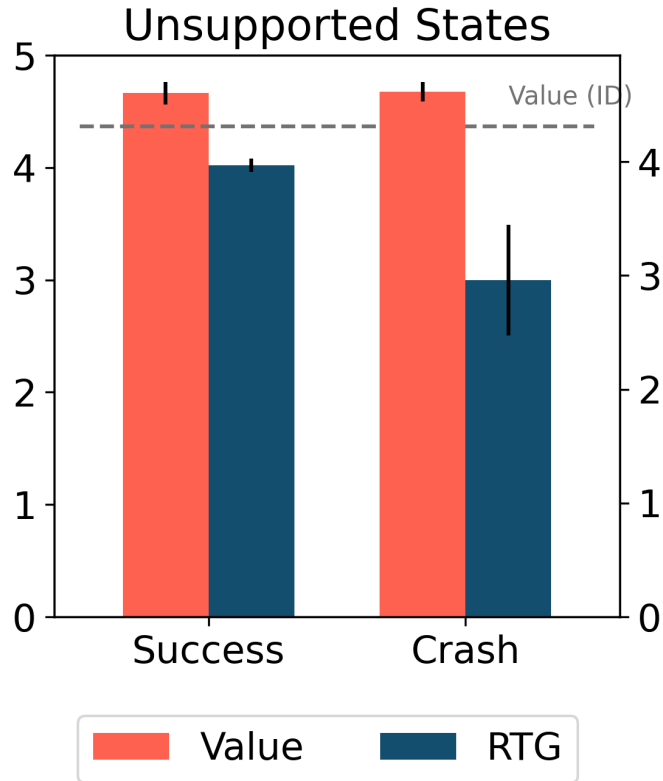


Figure A.1: Value prediction of PPO and actual return-to-go (RTG) in unsupported states in Roundabout environment

## **A.4 Experiment Details**

### **A.4.1 Hardware and Platform**

Experiments are run on a single NVIDIA RTX A6000 GPUs, with all code implemented in PyTorch.

### **A.4.2 Statistics**

All mean value is obtained by running with three different seeds and calculated with `numpy.mean()`. All error bar is obtained by `numpy.std()`.

### **A.4.3 Model Architecture**

The backbone for Unet is based on [34]. We add cross-attention blocks after each residual block, except for the bottleneck layers. Inputs to the cross-attention blocks are the conditioning embedding and output of the residual block. To ensure local consistency of trajectory, we used 1D convolution along the horizon dimension. To keep the number of parameters for cross attention and the original Unet relatively balanced, we also used 1D convolution as the mapping from input to key, query, and value. Detailed model architecture is shown below in Figure [A.2](#).

The number of down-sampling/up-sampling and feature channel sizes is different for each experiment. Detailed parameters can be found in the section for each experiment.

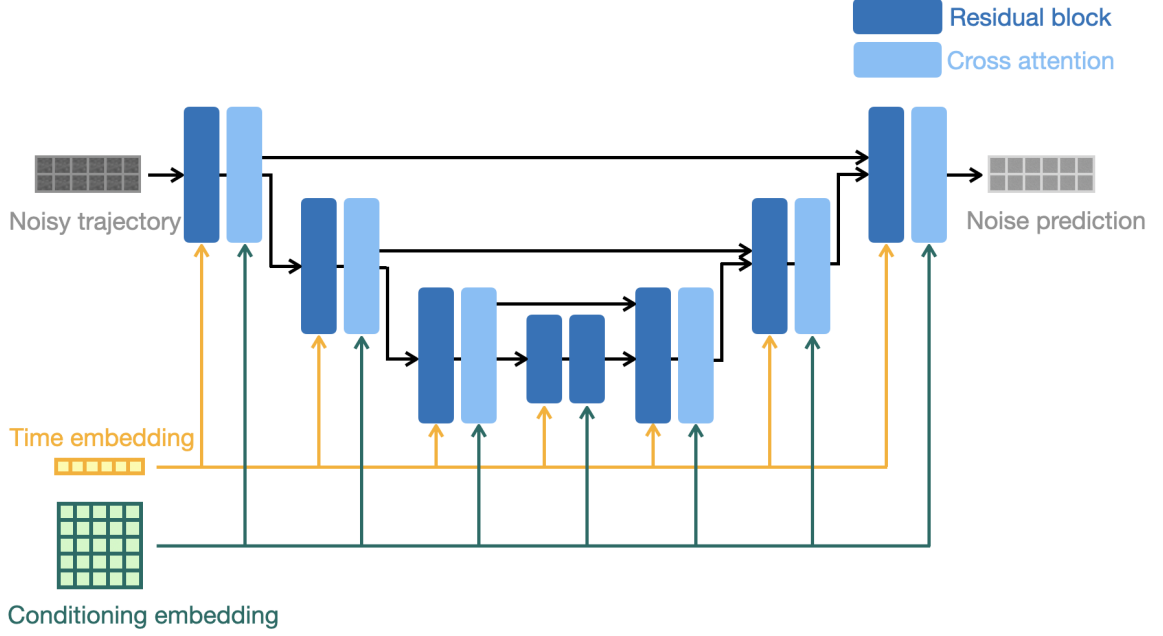


Figure A.2: Model architecture.

#### A.4.4 Trajectory formulation

The trajectory  $\tau \in \mathbb{R}^d$  is represented by concatenating the state  $\mathbf{s}_u \in \mathbb{R}^{d_s}$  and the action  $\mathbf{a}_u \in \mathbb{R}^{d_A}$  at planning time step  $u$  and then horizontally stacking them for all time steps. For example, a trajectory with planning horizon  $h$  can be written as

$$\tau = \begin{bmatrix} \mathbf{s}_1 & \mathbf{s}_2 & \dots & \mathbf{s}_h \\ \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_h \end{bmatrix}.$$

#### A.4.5 Pseudo-code

Pseudo-code for planning with conditional diffusion model is shown below in Algorithm 1.

---

**Algorithm 1** Planning with Attention-based Composition Conditioned Diffusion Model

---

**Input:** Diffusion model  $\epsilon_\theta$ , compositional elements extractor  $r$ , learnable embedding function  $h$ , classifier-free guidance scale  $\lambda$ , state dimensionality  $d_S$ , initial observation  $\mathbf{o}$ , environment simulator  $env$

**while** not done **do**

Initialize  $\tau_t \sim \mathcal{N}(0, I)$

$\mathbf{c} \leftarrow r(\mathbf{o})$  ▷ Extract observed compositional information

$\mathbf{z} \leftarrow h(\mathbf{c})$  ▷ Obtain element embedding

**for**  $t \leftarrow T, \dots, 1$  **do**

$\tau_t[:d_S, 0] \leftarrow \sqrt{\bar{\alpha}_t} \mathbf{o} + \sqrt{1 - \bar{\alpha}_t} \epsilon$ , where  $\epsilon \sim \mathcal{N}(0, I)$  ▷ Replace the first observed

$\tilde{\epsilon}_t = (1 + \lambda) \epsilon_\theta(\mathbf{s}_t, \mathbf{z}, t) - \lambda \epsilon_\theta(\mathbf{s}_t, t)$  state with noised  $\mathbf{o}$   
▷ Classifier free guidance

$\tau_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \tau_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \tilde{\epsilon}_t \right) + \sigma_t \epsilon_t$ , where  $\epsilon_t \sim \mathcal{N}(0, I)$

**end for**

$\mathbf{a} \leftarrow \tau_0[d_S :, 0]$  ▷ Extract action

$\mathbf{o} \leftarrow env.step(\mathbf{a})$

**end while**

---

#### A.4.6 Maze2D

##### Extra Results

##### Experiment Details

We followed the setup used in [34]. The hyperparameters shared for large and medium mazes are shown below in Table A.1. Large maze use a planning horizon of 384 and medium maze use a planning horizon of 256. Conditioning is passed through a positional embedding layer first to map each dimension of the waypoint  $(x, y, v_x, v_y)$  to a higher dimension of 21 and concatenate them to form a vector of size  $(1, 21 * 4)$ . Three waypoints are then stacked together to form a matrix of size  $(3, 21 * 4)$  and passed into the cross-attention layer. In our experiment, directly using the waypoints as conditioning was unsuccessful.

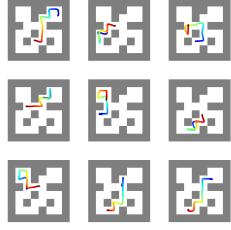


Figure A.3: train traj

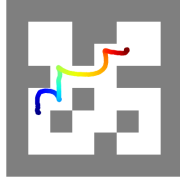


Figure A.4: ID (Unconditioned)

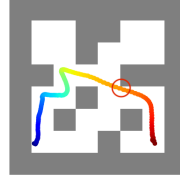


Figure A.5: OOD (Unconditioned)

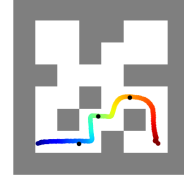


Figure A.6: cfg=1.3 (Conditioned)

Figure A.7: Trajectories generated in Maze2D for medium maze. (a) are samples from the training set. (b) are trajectories generated by the unconditioned diffusion model given in distribution start and end positions. (c) are generated by the unconditioned diffusion model on unseen start and end positions. (d) are generated by a conditioned diffusion model using 3 waypoints (black dots) as conditioning with classifier-free guidance (cfg) weight 1.3.

#### A.4.7 Roundabout

##### Environment

The training environment consists of all cars or all bicycles and the testing environment is a mixture of traffic (Figure A.12).

##### Environment Parameters

We changed the parameters to create a different type of traffic in the roundabout as shown below in Table A.4. Also, since bicycles have slower speeds, we change the initialization position so that each environment vehicle can interact with the ego vehicle.

| Parameter  | Value     |
|--|-----------|
| number of diffusion steps                              | 256       |
| action weight  | 1         |
| dimension multipliers                                  | (1, 4, 8) |
| classifier free guidance drop conditioning probability | 0.1       |
| steps per epoch  | 10000     |
| loss type  | l2        |
| train steps  | 2e6       |
| batch size   | 32        |
| learning rate  | 2e-4      |
| gradient accumulate every                              | 2         |
| ema decay  | 0.995     |

Table A.1: Training parameter for diffusion model in Maze2D

| Parameter                | Car      | Bicycle |
|--------------------------|----------|---------|
| length                   | 5.0      | 2.0     |
| width                    | 2.0      | 1.0     |
| speed                    | [23, 25] | 4       |
| max acceleration         | 6.0      | 2.0     |
| comfort max acceleration | 3.0      | 1.0     |

Table A.2: Parameters for car and bicycles in Roundabout environment

## Dataset

In order to collect expert trajectories, we train two PPO models separately on the environment with all cars and all bicycles. We then collect 320000 successful trajectories in the training environment. All trajectories have a unified length of 12.

### A.4.8 Setup for Roundabout Environment

Here we describe how the Roundabout task in this paper conforms to our problem description. In this setting our base object set is  $E = \{\text{car}, \text{bicycle}, \text{null}\}$  where null means an object is non-visible. Since the maximum number of objects in the roundabout is five and we fix the ego agent to be a car, support for the training observation is

$\{(\text{car (ego agent)}, \text{car}, \text{car}, \text{car}, \text{car}), (\text{car (ego agent)}, \text{bicycle}, \text{bicycle}, \text{bicycle}, \text{bicycle})\}$  and for the testing observation is  $\{(\text{car (ego agent)}, \text{bicycle}, \text{bicycle}, \text{car}, \text{car})\}$  assuming no ordering and when the state is fully observable. Since the supports for training and testing are non-overlapping under full observability, they will remain non-overlapping even when some traffic objects are out of sight, unless the ego agent is the only object present in the environment.

## Experiment Details

We use `stable_baseline3` [65] as the implementation for PPO. The parameter is the default parameter used in the Highway environment [45]. We increased total timesteps because the environment now has two modalities (all cars and all bicycles) and we observed that PPO takes longer to converge. Detailed parameters for PPO and diffusion are shown below in Table A.3 and Table A.4.

| Parameter       | Value     |
|-----------------|-----------|
| policy          | MlpPolicy |
| batch size      | 64        |
| n_steps         | 768       |
| n_epochs        | 10        |
| learning rate   | 5e-4      |
| gamma           | 0.8       |
| total timesteps | 2e5       |

Table A.3: Training parameter for PPO

| Parameter  | Value     |
|--|-----------|
| planning horizon                                       | 8         |
| number of diffusion steps                              | 80        |
| action weight  | 10        |
| dimension multipliers                                  | (1, 4, 8) |
| conditioning embedding size                            | 20        |
| classifier free guidance drop conditioning probability | 0.1       |
| classifier free guidance weight                        | 1.0       |
| steps per epoch  | 10000     |
| loss type  | l2        |
| train steps  | 1e4       |
| batch size   | 32        |
| learning rate  | 2e-4      |
| gradient accumulate every                              | 2         |
| ema decay  | 0.995     |

Table A.4: Training parameter for diffusion model in Roundabout

## Model Size

We include the model size for different algorithms below in Table A.5. To eliminate the concern for performance gain due to model size, we include the performance of a large BC model and PPO that has roughly the same number of parameters as the conditioned diffusion model.

|                      | BC          | PPO         | Diffusion  | Large BC   | Large PPO                                   |
|----------------------|-------------|-------------|------------|------------|---|
| Model size           | 0.30 MB     | 0.60 MB     | 54.19 MB   | 55.65 MB   | 111.29MB (Policy:55.64+Value:55.64)         |
| Number of parameters | 75013       | 148998      | 13546370   | 13912325   | 27823622 (Policy:13911040 + Value:13911040) |
| OOD reward           | 7.50 (0.03) | 8.19 (0.16) | 8.81 (0.2) | 7.71 (0.3) | 8.43 (0.19)                                 |
| OOD crashes          | 37.7 (0.5)  | 36.0 (5.7)  | 20.0 (2.5) | 37.3 (4.0) | 31.67 (1.89)                                |

Table A.5: Model size, number of parameters, and performance for different models.

### **Reliable Conditioning**

We demonstrate the importance of having reliable information of base element composition, we compare the performance of the conditioned diffusion model given random and ground truth conditionings.

|                   | <b>Ground Truth</b> | <b>Random</b> |
|-------------------|---------------------|---------------|
| Number of Crashes | 19.67 (2.49)        | 24.33 (0.47)  |
| Reward            | 8.81(0.2)           | 8.1 (0.09)    |

Table A.6: Performance of conditioned diffusion model given ground truth and random conditionings.

### A.4.9 StarCraft

#### Experiment Details

We use the codebase OpenRL [30] for the implementation of MAPPO. Detailed parameters for MAPPO can be found in Table A.7.

| Parameter                 | Value |
|---------------------------|-------|
| learning rate actor       | 5e-4  |
| learning rate critic      | 1e-3  |
| data chunk length         | 8     |
| env num                   | 8     |
| episode length            | 400   |
| PPO epoch                 | 5     |
| actor train interval step | 1     |
| use recurrent policy      | True  |
| use adv normalize         | True  |
| use value active masks    | False |
| use linear LR decay       | True  |

Table A.7: MAPPO hyper-parameters used for SMACv2. We utilize the hyperparameters used in SMACv2 [17].

Detailed parameters for training a conditioned diffusion model for 5v5 are shown below in Table A.8 and 3v3 in Table A.9.

| <b>Parameter</b>                                       | <b>Value</b>         |
|--|----------------------|
| planning horizon                                       | 40                   |
| number of diffusion steps                              | 256                  |
| action weight  | 1                    |
| dimension multipliers                                  | (1, 4, 8)            |
| conditioning embedding size                            | 40                   |
| classifier free guidance drop conditioning probability | 0.1                  |
| classifier free guidance weight                        | [0.7, 1.0, 1.3, 1.5] |
| steps per epoch  | 10000                |
| loss type  | l2                   |
| train steps  | 2e6                  |
| batch size   | 32                   |
| learning rate  | 2e-4                 |
| gradient accumulate every                              | 2                    |
| ema decay  | 0.995                |

Table A.8: Training parameter for diffusion model in StarCraft for 5v5

| <b>Parameter</b>                                       | <b>Value</b>         |
|--|----------------------|
| planning horizon                                       | 32                   |
| number of diffusion steps                              | 256                  |
| action weight  | 1                    |
| dimension multipliers                                  | (1, 4, 8)            |
| conditioning embedding size                            | 40                   |
| classifier free guidance drop conditioning probability | 0.1                  |
| classifier free guidance weight                        | [0.7, 1.0, 1.3, 1.5] |
| steps per epoch  | 10000                |
| loss type  | l2                   |
| train steps  | 2e6                  |
| batch size   | 32                   |
| learning rate  | 2e-4                 |
| gradient accumulate every                              | 2                    |
| ema decay  | 0.995                |

Table A.9: Training parameter for diffusion model in StarCraft for 3v3

## Dataset Initial State Distribution

The probability of generating each unit type in SMACv2 is imbalanced. Specifically, the probability for Stalker, Zealot, and Colossus is 0.45, 0.45, and 0.1 respectively. The initial state distribution of training trajectories collected by MAPPO for random combination is shown below in Figure A.13 and A.14. Since we only keep the successful trajectories and use them as expert data, the distribution depends on the generation probability and MAPPO success rate for different team combinations. A total number of 240000 trajectories were used to train the diffusion model. Since diffusion is trained on local observations and actions of all MAPPO actors, the total number of training samples is  $5 \times 240000$  for 5v5 and  $3 \times 240000$  for 3v3.

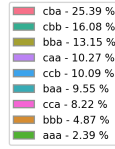


Figure A.13: Distribution of initial state for 3v3 simple scenario

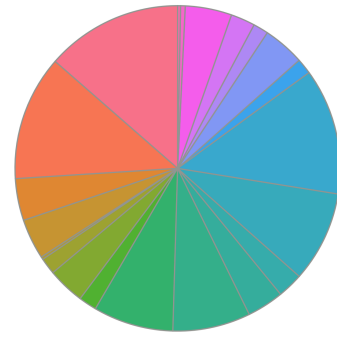
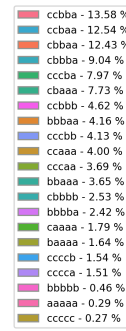


Figure A.14: Distribution of initial state for 5v5 simple scenario

### Detailed Results on SMACv2

Table A.10 and A.11 show the detailed performance of different algorithms in the 3v3 and 5v5 scenarios, respectively.

| Env:<br>3v3                    | RL             |                    | Imitation Learning |                     |
|--------------------------------|----------------|--------------------|--------------------|---------------------|
|                                | 2 PPO + 1 Rand | 3 PPO              | BC                 | 2 PPO + 1 Diffusion |
| $ABC \rightarrow ABC$ (ID)     | 0.18 (0.01)    | 0.58 (0.02)        | 0.58 (0.07)        | <b>0.59 (0.04)</b>  |
| $ABC \rightarrow AAA$ (Simple) | 0.07 (0.03)    | 0.52 (0.03)        | 0.56 (0.02))       | <b>0.59 (0.02)</b>  |
| $AAA \rightarrow AAA$ (ID)     | 0.09 (0.04)    | <b>0.63 (0.02)</b> | 0.6 (0.02)         | 0.61 (0.05)         |
| $AAA \rightarrow ABC$ (Hard)   | 0.11 (0.02)    | 0.42 (0.02)        | 0.4 (0.06)         | <b>0.49(0.02)</b>   |

Table A.10: Success rate of each agent in 100 rounds. The first two rows correspond to the simple setting of generalization to states with different support and the last two rows correspond to non-overlapping support. Numbers in the parenthesis represent the standard error over 3 seeds. The best performing method is labeled bold. The 2 PPO + 1 Rand column shows the effect of replacing one PPO trained agent with a random agent as a baseline for comparison against the 2 PPO + 1 Diffusion case.

| Env:<br>5v5                    | RL             |                    | Imitation Learning |                     |
|--------------------------------|----------------|--------------------|--------------------|---------------------|
|                                | 4 PPO + 1 Rand | 5 PPO              | BC                 | 4 PPO + 1 Diffusion |
| $ABC \rightarrow ABC$ (ID)     | 0.22 (0.04)    | 0.64 (0.05)        | 0.56 (0.05)        | <b>0.66 (0.01)</b>  |
| $ABC \rightarrow AAA$ (Simple) | 0.11 (0.03)    | 0.54 (0.04)        | 0.52 (0.05)        | <b>0.56 (0.02)</b>  |
| $AAA \rightarrow AAA$ (ID)     | 0.14 (0.02)    | <b>0.58 (0.04)</b> | 0.54 (0.04)        | 0.55 (0.03)         |
| $AAA \rightarrow ABC$ (Hard)   | 0.11 (0.02)    | 0.26 (0.05)        | 0.28 (0.04)        | <b>0.32 (0.04)</b>  |

Table A.11: Success rate of each agent in 100 rounds. The first two rows correspond to the simple setting of generalization to states with different support and the last two rows correspond to non-overlapping support. Numbers in the parenthesis represent the standard error over 3 seeds. The best performing method is labeled bold. The 4 PPO + 1 Rand column shows the effect of replacing one PPO trained agent with a random agent as a baseline for comparison against the 4 PPO + 1 Diffusion case.

### Detailed Results for Ablation

The ablation result for 3v3 and 5v5 scenarios are shown below in Table A.12 and Table A.13. The first column is the success rate without conditioning (No Cond).

The second column represents concatenating the conditioning with time embedding (Concat). The last column represents passing conditioning as another input beside the trajectory to the cross-attention block (Attn).

Table A.12: Ablation for Diffusion on 3v3

| Env 3v3                        | 2 PPO + 1 Diffusion |           |           |
|--------------------------------|---------------------|-----------|-----------|
|                                | No Cond             | Concat    | Attn      |
| $ABC \rightarrow ABC$ (ID)     | 0.55±0.03           | 0.59±0.04 | 0.59±0.05 |
| $ABC \rightarrow AAA$ (Simple) | 0.5±0.06            | 0.59±0.02 | 0.59±0.02 |
| $AAA \rightarrow AAA$ (ID)     | 0.4±0.03            | 0.64±0.03 | 0.61±0.05 |
| $AAA \rightarrow ABC$ (Hard)   | 0.28±0.03           | 0.44±0.05 | 0.49±0.02 |

Table A.13: Ablation for Diffusion on 5v5

| Env 5v5                        | 4PPO + 1 Diffusion |           |           |
|--------------------------------|--------------------|-----------|-----------|
|                                | No Cond            | Concat    | Attn      |
| $ABC \rightarrow ABC$ (ID)     | 0.53±0.04          | 0.59±0.03 | 0.66±0.01 |
| $ABC \rightarrow AAA$ (Simple) | 0.50±0.03          | 0.50±0.01 | 0.56±0.02 |
| $AAA \rightarrow AAA$ (ID)     | 0.47±0.08          | 0.55±0.03 | 0.58±0.04 |
| $AAA \rightarrow ABC$ (Hard)   | 0.27±0.03          | 0.32±0.04 | 0.30±0.04 |

## 2v2

The success rates for StarCraft 2v2 are shown below in Table A.14. We can see that out-of-combination cases did not cause the performance to drop drastically for MAPPO. This is because the number of combinations in 2v2 is very limited (e.g.  $aa, bb, ab$ ), and if one agent dies, MAPPO has encountered scenarios of playing with each unit type individually, therefore falling back to in distribution state again. This scenario also exists for 5v5 and 3v3 but only at the end of each game when only one agent is left.

Table A.14: SMAC II success rate for 2v2

| Env                            | BC              | MAPPO           |                                 | Diffusion       |                                 |                                 |
|--------------------------------|-----------------|-----------------|---------------------------------|-----------------|---------------------------------|---------------------------------|
|                                | BC              | 1 PPO + 1 Rand  | 5 PPO                           | No Cond         | Concat                          | Attn                            |
| $ABC \rightarrow ABC$ (ID)     | 0.54 $\pm$ 0.02 | 0.06 $\pm$ 0.02 | <b>0.62<math>\pm</math>0.05</b> | 0.43 $\pm$ 0.04 | 0.56 $\pm$ 0.03                 | 0.57 $\pm$ 0.067                |
| $ABC \rightarrow AAA$ (Simple) | 0.47 $\pm$ 0.02 | 0.02 $\pm$ 0.01 | <b>0.57<math>\pm</math>0.02</b> | 0.44 $\pm$ 0.02 | 0.55 $\pm$ 0.06                 | 0.49 $\pm$ 0.06                 |
| $AAA \rightarrow AAA$ (ID)     | 0.57 $\pm$ 0.01 | 0.01 $\pm$ 0.01 | <b>0.64<math>\pm</math>0</b>    | 0.38 $\pm$ 0.02 | <b>0.63<math>\pm</math>0.04</b> | <b>0.63<math>\pm</math>0.02</b> |
| $AAA \rightarrow ABC$ (Hard)   | 0.4 $\pm$ 0.03  | 0.04 $\pm$ 0.02 | <b>0.44<math>\pm</math>0.02</b> | 0.29 $\pm$ 0.02 | <b>0.43<math>\pm</math>0.08</b> | 0.41 $\pm$ 0.06                 |

### More Rendering of States Predicted by the Diffusion model

More rendering of the future states predicted by the diffusion model is shown in Figure A.15.

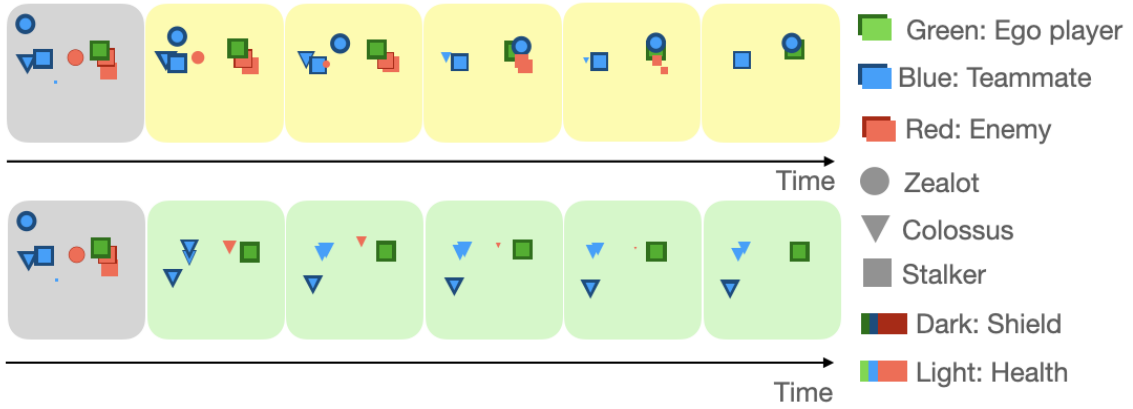


Figure A.15: Rendering of future states predicted by the diffusion model given different conditionings. The grey box is the initial state. Yellow boxes are conditioned on the type of unit in the initial state. Green boxes are conditioned on all Triangles. Smaller sizes represent less shield or health.

#### A.4.10 Model Runtime and GPU Memory

We include the training time and GPU memory used for the conditioned diffusion model below in Table A.15 and A.16.

| Training Time | Roundabout | SMACv2 2v2 | SMACv2 3v3 | SMACv2 5v5 |
|---------------|------------|------------|------------|------------|
| PPO           | 0.5h       | 9h         | 9h         | 9h         |
| Diffusion     | 1h         | 48h        | 70h        | 98h        |

Table A.15: Training time for PPO and conditioned diffusion model in different environments.

| GPU Memory | Roundabout | SMACv2 2v2 | SMACv2 3v3 | SMACv2 5v5 |
|------------|------------|------------|------------|------------|
| Diffusion  | 542 MiB    | 1004 MiB   | 2892 MiB   | 4096 MiB   |

Table A.16: GPU Memory for training conditioned diffusion model in different environments.

#### **A.4.11 Parameter Comparison with Concatenation or Attention**

We demonstrate the number of parameters in attention-based conditioning and concatenation-based conditioning to eliminate the concern regarding performance gain due to more parameters. Attention or concatenation has roughly the same number of parameters as the attention module is convolutional layers and concatenation increases the parameters of conditioning layer.

|            | <b>Attention</b> | <b>Concatenation</b> |
|------------|------------------|----------------------|
| Model size | 617.06 MB        | 619.64 MB            |
| Parameters | 154264085        | 154911187            |

Table A.17: Number of parameters in attention-based conditioning and concatenation-based conditioning.

#### **A.4.12 Substituting More MAPPO Agents with Diffusion Agents**

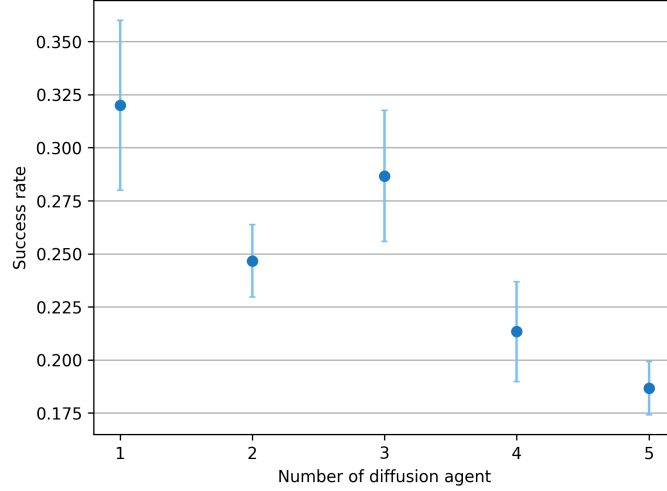


Figure A.16: Success rate vs number of agents in SMACv2 5v5 hard scenario that are replaced with diffusion agents. Replacing more than one MAPPO agent with diffusion agents hurts performance.

We would like to ask the question of what about replacing more than one MAPPO agent with diffusion model. Figure A.16 shows that the number of diffusion models does not have a positive correlation with the success rate. This is because MAPPO can learn a collaborative policy between actors and simply adding more ego-centric diffusion models will break the coordination between actions. Also, since the diffusion model is trained to play with all PPOs teammates, replacing other PPO actions with actions generated by diffusion models will cause a distribution shift that is hard to quantify.

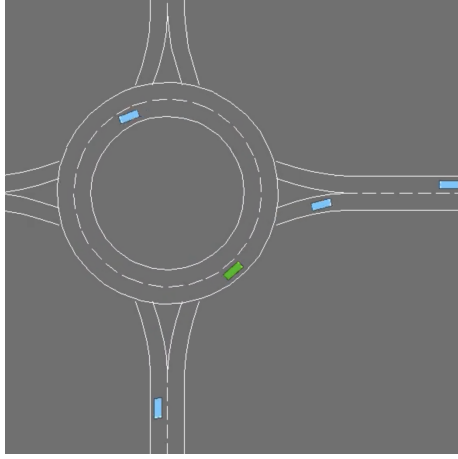


Figure A.8: train env 1 (cars)

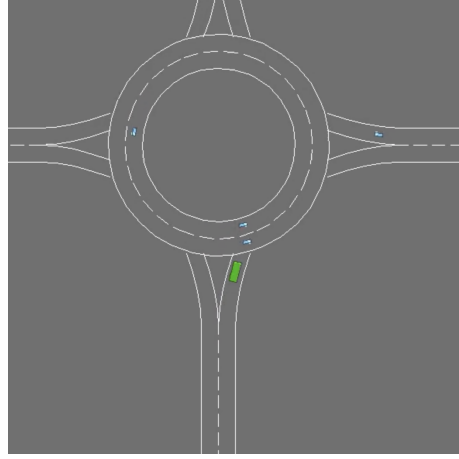


Figure A.9: train env 2 (bikes)

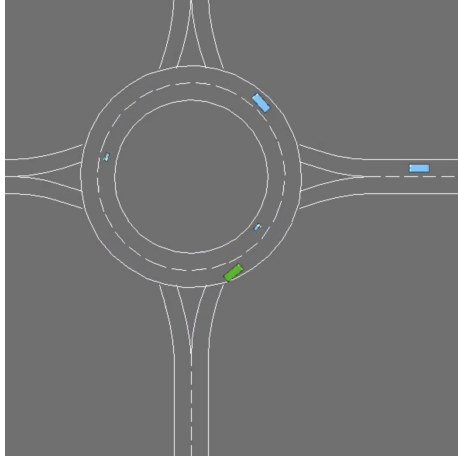


Figure A.10: test env 1

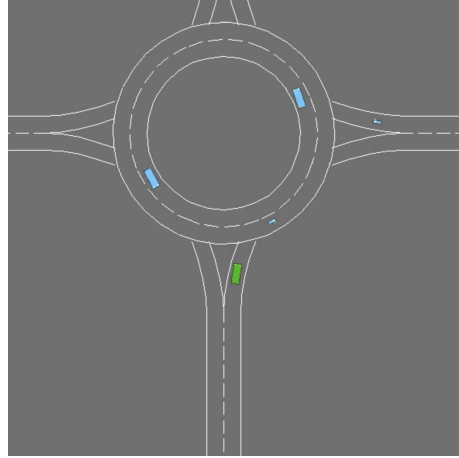


Figure A.11: test env 2

Figure A.12: Training and testing environments for Roundabout. The green vehicle is the ego agent and the blue ones are controlled by the environment. The large blue box represents a car and the small blue box represents a bicycle.



# Appendix B

## Accelerating Diffusion Models in Offline RL via Reward-Aware Consistency Trajectory Distillation

### B.1 Model Architecture

We follow the model architecture used in Chi et al. [9] and Prasad et al. [64] and continue to use the 1D temporal CNN layer for our Unet and FiLM layers to process the conditioning information.

#### B.1.1 Model Sizes for Maze2d

Model parameters for teacher models of Umaze, Medium, and Large Maze are shown below in Table B.1. The student model has the same architecture as the teacher model except it also takes one extra variable of denoising timestep as conditioning.

#### B.1.2 Model Sizes for Gym-MuJoCo

Unet parameters for teacher and student models in the MuJoCo task are shown below in Table B.2. Model sizes are fixed through 9 different environments.

We follow the setup in Janner et al. [34], where we use Linear layers and Mish

| <b>Parameter</b>         | <b>Umaze</b>     | <b>Medium</b>     | <b>Large</b>           |
|--------------------------|------------------|-------------------|------------------------|
| diffusion_step_embed_dim | 256              | 256               | 256                    |
| down dims                | [256, 512, 1024] | [512, 1024, 2048] | [256, 512, 1024, 2048] |
| horizon                  | 128              | 256               | 384                    |
| kernel size              | 5                | 5                 | 5                      |

Table B.1: Model parameters for Unet in Maze2d.

| <b>Parameter</b>         | <b>MuJoCo</b>     |
|--------------------------|-------------------|
| diffusion_step_embed_dim | 128               |
| down dims                | [512, 1024, 2048] |
| horizon                  | 16                |
| kernel size              | 5                 |

Table B.2: Model parameters for Unet in MuJoCo.

layers [58] for the reward model. Reward model architecture remains the same across all MuJoCo benchmarks. Model parameters are shown below in Table B.3.

## B.2 Training Details

Our models are trained on D4RL dataset [20], which was released under Apache-2.0 license.

### B.2.1 Noise Scheduler

We follow the setup in Prasad et al. [64] and use EDM noise scheduler [39] for the teacher model. Particularly, discretization bins are chosen to be 80.

Student model used CTM scheduler [41] also with discretization bins of 80.

### B.2.2 Reward Discount Factor

We use  $\gamma = 0.997$  for hopper,  $\gamma = 0.99$  for walker2d, and  $\gamma = 0.9$  for halfcheetah, along with an early termination penalty of -100.

| <b>Parameter</b> | <b>MuJoCo</b>      |
|------------------|--------------------|
| layer dimensions | [32, 64, 128, 256] |

Table B.3: Reward model parameters in MuJoCo.

| <b>Parameter</b>          | <b>CTM</b> | <b>DSM</b> | <b>Reward</b> |
|---------------------------|------------|------------|---------------|
| hopper-medium-replay      | 1.0        | 1.0        | 1.0           |
| hopper-medium             | 1.0        | 1.0        | 3.0           |
| hopper-medium-expert      | 1.0        | 1.0        | 0.0           |
| walker2d-medium-replay    | 1.0        | 1.0        | 1.0           |
| walker2d-medium           | 1.0        | 1.0        | 0.4           |
| walker2d-medium-expert    | 1.0        | 1.0        | 0.2           |
| halfcheetah-medium-replay | 1.0        | 1.0        | 1.0           |
| halfcheetah-medium        | 1.0        | 1.0        | 0.5           |
| halfcheetah-medium-expert | 1.0        | 1.0        | 0.0           |

Table B.4: Weights for CTM, DSM, and Reward loss used in MuJoCo benchmark.

### B.2.3 Weight of Different Losses

The weights for CTM, DSM, and Reward loss we used in the experiment are shown below in Table B.4. Generally, if the training dataset includes more expert samples, the weight for reward guidance is smaller. A reward weight of 0.0 resembles behavior cloning with consistency trajectory distillation. We found that as long as the loss weights are chosen to keep the individual loss terms within the same order of magnitude, the model will achieve reasonable performance.

## B.3 More Ablations

### B.3.1 walker-medium

Table B.5: We compare incorporating the reward model in different stages of training on MuJoCo walker-medium. Results are presented as the mean and standard error across 100 seeds.

| <b>Walker<br/>Medium</b> | Unconditioned<br>teacher | Reward-Aware<br>teacher |
|--------------------------|--------------------------|-------------------------|
| Unconditioned student    | 93.3 $\pm$ 1.8           | 97.0 $\pm$ 1.0          |
| Reward-Aware student     | <b>98.9</b> $\pm$ 1.6    | 94.5 $\pm$ 2.6          |

### B.3.2 Comparing with fast sampling algorithms for Maze2d

Table B.6: We compare fast sampling algorithms DDIM and CTD, along with our method RACTD on Maze2d environment. DDIM performs fast sampling based on a DDPM model, while CTD and RACTD (ours) distill an EDM teacher. The number of function evaluations (NFE) reflects the sampling speed of each algorithm. Results are reported as the mean and standard error over 100 random seeds.

| Method       | NFE       | U-Maze                 | Medium                 | Large                  | Average Score |
|--------------|-----------|------------------------|------------------------|------------------------|---------------|
|              |           | Score                  | Score                  | Score                  |               |
| DDPM         | 100       | <b>126.3</b> $\pm$ 0.7 | <u>126.8</u> $\pm$ 3.0 | <u>144.8</u> $\pm$ 4.9 | <u>132.6</u>  |
| DDIM         | <u>10</u> | 121.2 $\pm$ 1.1        | 126.2 $\pm$ 2.8        | 143.1 $\pm$ 4.9        | 130.2         |
| DDIM         | <b>1</b>  | 3.5 $\pm$ 4.7          | -2.6 $\pm$ 12.5        | -1.5 $\pm$ 0.7         | -0.2          |
| EDM          | 80        | 125.4 $\pm$ 0.6        | 120.1 $\pm$ 4.2        | <b>149.0</b> $\pm$ 0.5 | 131.5         |
| CTD          | <b>1</b>  | 123.4 $\pm$ 1.0        | 119.8 $\pm$ 4.1        | 127.1 $\pm$ 6.4        | 123.4         |
| RACTD (ours) | <b>1</b>  | <u>125.7</u> $\pm$ 0.6 | <b>130.8</b> $\pm$ 1.8 | 143.8 $\pm$ 0.0        | <b>133.4</b>  |

## B.4 The Trade-off between Mode Selection and Sample Diversity

In this section, we include a discussion about the trade-off between mode selection induced by our reward-aware training and the sample diversity of the student. Naturally, favoring selected modes can lead to generation with limited sample diversity as summarized in Table B.7. This trade-off between sample diversity and sample optimality observed in RACTD is similar to what has been seen in other generative domains (e.g., language model RLHF [29], classifier-guided diffusion, conditional image generation), where preference alignment also often reduces sample diversity. In our case, the reward model acts similarly to a classifier or an alignment reward model, guiding the model toward desirable behaviors and sacrificing some of the sample diversity by design.

Importantly, our decoupled framework allows the use of a single, unconditioned teacher with strong generalization capabilities across tasks. For multi-task or unseen-task settings, different reward models can be trained per task, and corresponding student models can be distilled from the same teacher using different reward models.

Table B.7: A summarization of the trade off between sample diversity and model performance.

|  | Sample diversity | Performance | Sample time |
|--|------------------|-------------|-------------|
| Reward agnostic diffusion                | High             | Low         | Slow        |
| Reward aware diffusion                   | Low              | High        | Slow        |
| Reward agnostic consistency distillation | High             | Low         | Fast        |
| Reward aware consistency distillation    | Low              | High        | Fast        |

## B.5 Limitations and Future work

One limitation of our approach is the need to train three separate networks: the teacher, student, and reward model. Training the teacher can be time-consuming, as achieving strong performance often requires a higher number of denoising steps. Additionally, consistency trajectory distillation is prone to loss fluctuations, and

incorporating a reward model into the distillation process may further amplify this instability. Future work will focus on developing a more stable and efficient training procedure, as well as exploring methods to integrate non-differentiable reward models into the framework.

# Bibliography

- [1] Ossama Ahmed, Frederik Träuble, Anirudh Goyal, Alexander Neitz, Yoshua Bengio, Bernhard Schölkopf, Manuel Wüthrich, and Stefan Bauer. Causalworld: A robotic manipulation benchmark for causal structure and transfer learning. *arXiv preprint arXiv:2010.04296*, 2020. [A.1.1](#)
- [2] Sumukh K Aithal, Pratyush Maini, Zachary C Lipton, and J Zico Kolter. Understanding hallucinations in diffusion models through mode interpolation. *arXiv preprint arXiv:2406.09358*, 2024. [2.4.2](#)
- [3] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022. [2.5.2](#), [3.1](#), [3.3.1](#), [3.6](#)
- [4] Arthur Argenson and Gabriel Dulac-Arnold. Model-based offline planning. *arXiv preprint arXiv:2008.05556*, 2020. [3.4.1](#)
- [5] David Berthelot, Arnaud Autef, Jierui Lin, Dian Ang Yap, Shuangfei Zhai, Siyuan Hu, Daniel Zheng, Walter Talbott, and Eric Gu. Tract: Denoising diffusion models with transitive closure time-distillation. *arXiv preprint arXiv:2303.04248*, 2023. [3.6](#)
- [6] Adam Block, Ali Jadbabaie, Daniel Pfrommer, Max Simchowitz, and Russ Tedrake. Provable guarantees for generative behavior cloning: Bridging low-level stability and high-level behavior. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL <https://openreview.net/forum?id=PhFVF0gwid>. [3.1](#)
- [7] 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. [3.4.1](#)
- [8] Yuhui Chen, Haoran Li, and Dongbin Zhao. Boosting continuous control with consistency policy. *arXiv preprint arXiv:2310.06343*, 2023. [3.1](#), [3.6](#)
- [9] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via

- action diffusion. *arXiv preprint arXiv:2303.04137*, 2023. [3.1](#), [3.3.1](#), [3.3.2](#), [3.6](#), [B.1](#)
- [10] Hyungjin Chung, Suhyeon Lee, and Jong Chul Ye. Fast diffusion sampler for inverse problems by geometric decomposition. *arXiv preprint arXiv:2303.05754*, 2023. [2.1](#)
- [11] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International conference on machine learning*, pages 1282–1289. PMLR, 2019. [A.1.1](#)
- [12] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020. [A.1.1](#)
- [13] Michael Dennis, Natasha Jaques, Eugene Vinitzky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020. [A.1.1](#)
- [14] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021. [3.3.4](#)
- [15] Zihan Ding and Chi Jin. Consistency models as a rich and efficient policy class for reinforcement learning. *arXiv preprint arXiv:2309.16984*, 2023. [3.1](#), [3.3.1](#), [3.4.1](#), [3.6](#)
- [16] Xintong Duan, Yutong He, Fahim Tajwar, Wen-Tse Chen, Ruslan Salakhutdinov, and Jeff Schneider. State combinatorial generalization in decision making with conditional diffusion models. *arXiv preprint arXiv:2501.13241*, 2025. [3.1](#)
- [17] Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob N Foerster, and Shimon Whiteson. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2212.07489*, 2022. ([document](#)), [2.6.3](#), [A.7](#)
- [18] Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob Foerster, and Shimon Whiteson. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024. [2.6.3](#)
- [19] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017. [2.1](#), [A.1](#), [A.1.1](#)
- [20] 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. [2.7.1](#), [3.1](#), [3.3.5](#), [3.4.1](#), [3.4.2](#), [B.2](#)
- [21] Dibya Ghosh, Anurag Ajay, Pulkit Agrawal, and Sergey Levine. Offline rl policies should be trained to be adaptive. In *International Conference on Machine Learning*, pages 7513–7530. PMLR, 2022. [2.1](#)
  - [22] Jake Grigsby and Yanjun Qi. Measuring visual generalization in continuous control from pixels. *arXiv preprint arXiv:2010.06740*, 2020. [A.1.1](#)
  - [23] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019. [A.1.1](#)
  - [24] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018. [2.3](#), [2.3.1](#)
  - [25] Nicklas Hansen, Hao Su, and Xiaolong Wang. Stabilizing deep q-learning with convnets and vision transformers under data augmentation. *Advances in neural information processing systems*, 34:3680–3693, 2021. [A.1.1](#)
  - [26] Philippe Hansen-Estruch, Ilya Kostrikov, Michael Janner, Jakub Grudzien Kuba, and Sergey Levine. Idql: Implicit q-learning as an actor-critic method with diffusion policies. *arXiv preprint arXiv:2304.10573*, 2023. [3.3.1](#), [3.6](#)
  - [27] Yutong He, Naoki Murata, Chieh-Hsin Lai, Yuhta Takida, Toshimitsu Uesaka, Dongjun Kim, Wei-Hsiang Liao, Yuki Mitsufuji, J Zico Kolter, Ruslan Salakhutdinov, and Stefano Ermon. Manifold preserving guided diffusion. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=o3Bx0Loxm1>. [2.1](#)
  - [28] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. [2.4.1](#), [3.4.3](#), [3.6](#)
  - [29] Audrey Huang, Adam Block, Dylan J Foster, Dhruv Rohatgi, Cyril Zhang, Max Simchowitz, Jordan T Ash, and Akshay Krishnamurthy. Self-improvement in language models: The sharpening mechanism. *arXiv preprint arXiv:2412.01951*, 2024. [B.4](#)
  - [30] Shiyu Huang, Wentse Chen, Yiwen Sun, Fuqing Bie, and Wei-Wei Tu. Openrl: A unified reinforcement learning framework. *arXiv preprint arXiv:2312.16189*, 2023. [A.4.9](#)
  - [31] Geonho Hwang, Jaewoong Choi, Hyunsoo Cho, and Myungjoo Kang. Maganet: Achieving combinatorial generalization by modeling a group action. In *International Conference on Machine Learning*, pages 14237–14248. PMLR, 2023.

## A.1.2

- [32] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020. [A.1.1](#)
- [33] 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. [3.4.1](#)
- [34] Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022. [2.5.2](#), [2.6.1](#), [3.1](#), [3.3.1](#), [3.3.4](#), [3.4.1](#), [3.4.1](#), [3.4.2](#), [3.6](#), [A.4.3](#), [A.4.6](#), [B.1.2](#)
- [35] Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Ervin Teng, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. Obstacle tower: A generalization challenge in vision, control, and planning. *arXiv preprint arXiv:1902.01378*, 2019. [A.1.1](#)
- [36] Zahra Kadkhodaie, Florentin Guth, Eero P Simoncelli, and Stéphane Mallat. Generalization in diffusion models arises from geometry-adaptive harmonic representation. *arXiv preprint arXiv:2310.02557*, 2023. [2.4.2](#)
- [37] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 267–274, 2002. [2.1](#)
- [38] Bingyi Kang, Xiao Ma, Chao Du, Tianyu Pang, and Shuicheng Yan. Efficient diffusion policies for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024. [2.5.2](#)
- [39] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in neural information processing systems*, 35:26565–26577, 2022. [3.1](#), [3.2.2](#), [3.4.3](#), [3.6](#), [B.2.1](#)
- [40] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *Advances in neural information processing systems*, 33:21810–21823, 2020. [3.4.1](#)
- [41] Dongjun Kim, Chieh-Hsin Lai, Wei-Hsiang Liao, Naoki Murata, Yuhta Takida, Toshimitsu Uesaka, Yutong He, Yuki Mitsufuji, and Stefano Ermon. Consistency trajectory models: Learning probability flow ode trajectory of diffusion. *arXiv preprint arXiv:2310.02279*, 2023. [1](#), [3.1](#), [3.2.3](#), [3.6](#), [B.2.1](#)
- [42] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021. [3.4.1](#)
- [43] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information*

- Processing Systems*, 33:1179–1191, 2020. [2.1](#), [2.3](#), [3.4.1](#)
- [44] Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020. [A.1.1](#)
  - [45] Edouard Leurent. An environment for autonomous driving decision-making. <https://github.com/eleurent/highway-env>, 2018. [2.3.1](#), [2.6.2](#), [A.4.8](#)
  - [46] 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. [2.1](#), [2.3.1](#), [2.6.2](#)
  - [47] Haoran Li, Zhennan Jiang, Yuhui Chen, and Dongbin Zhao. Generalizing consistency policy to visual rl with prioritized proximal experience regularization. *arXiv preprint arXiv:2410.00051*, 2024. [3.1](#), [3.6](#)
  - [48] Jiachen Li, Quan Vuong, Shuang Liu, Minghua Liu, Kamil Ciosek, Henrik Christensen, and Hao Su. Multi-task batch reinforcement learning with metric learning. *Advances in Neural Information Processing Systems*, 33:6197–6210, 2020. [A.1](#)
  - [49] Haohe Liu, Zehua Chen, Yi Yuan, Xinhao Mei, Xubo Liu, Danilo Mandic, Wenwu Wang, and Mark D Plumbley. Audioldm: Text-to-audio generation with latent diffusion models. *arXiv preprint arXiv:2301.12503*, 2023. [3.1](#)
  - [50] Yuejiang Liu, Alexandre Alahi, Chris Russell, Max Horn, Dominik Zietlow, Bernhard Schölkopf, and Francesco Locatello. Causal triplet: An open challenge for intervention-centric causal representation learning. In *Conference on Causal Learning and Reasoning*, pages 553–573. PMLR, 2023. [A.1.2](#)
  - [51] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35:5775–5787, 2022. [3.1](#), [3.6](#)
  - [52] Guanxing Lu, Zifeng Gao, Tianxing Chen, Wenxun Dai, Ziwei Wang, and Yansong Tang. Manicm: Real-time 3d diffusion policy via consistency model for robotic manipulation. *arXiv preprint arXiv:2406.01586*, 2024. [3.1](#), [3.6](#)
  - [53] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11461–11471, 2022. [A.2](#)
  - [54] Eric Luhman and Troy Luhman. Knowledge distillation in iterative generative models for improved sampling speed. *arXiv preprint arXiv:2101.02388*, 2021. [3.6](#)

- [55] Jiafei Lyu, Xiaoteng Ma, Xiu Li, and Zongqing Lu. Mildly conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 35:1711–1724, 2022. [2.1](#)
- [56] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018. [A.1.1](#)
- [57] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022. [A.1.1](#)
- [58] Diganta Misra. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681*, 2019. [B.1.2](#)
- [59] 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. [A.1](#)
- [60] Milton Llera Montero, Casimir JH Ludwig, Rui Ponte Costa, Gaurav Malhotra, and Jeffrey Bowers. The role of disentanglement in generalisation. In *International Conference on Learning Representations*, 2020. [A.1.2](#)
- [61] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*, 2018. [A.1.1](#)
- [62] Maya Okawa, Ekdeep S Lubana, Robert Dick, and Hidenori Tanaka. Compositional abilities emerge multiplicatively: Exploring diffusion models on a synthetic task. *Advances in Neural Information Processing Systems*, 36, 2024. [2.1](#), [2.4.2](#), [A.1.2](#)
- [63] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018. [A.1.1](#)
- [64] Aaditya Prasad, Kevin Lin, Jimmy Wu, Linqi Zhou, and Jeannette Bohg. Consistency policy: Accelerated visuomotor policies via consistency distillation. *arXiv preprint arXiv:2405.07503*, 2024. [3.1](#), [3.2.2](#), [3.6](#), [B.1](#), [B.2.1](#)
- [65] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL <http://jmlr.org/papers/v22/20-1364.html>. [A.4.8](#)
- [66] Aravind Rajeswaran, Kendall Lowrey, Emanuel V Todorov, and Sham M Kakade.

- Towards generalization and simplicity in continuous control. *Advances in neural information processing systems*, 30, 2017. [A.1.1](#)
- [67] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR, 2019. [A.1](#), [A.1.1](#)
- [68] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=TIIdIXIpzhoI>. [3.1](#), [3.6](#)
- [69] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019. [2.6.3](#)
- [70] Lukas Schott, Julius Von Kügelgen, Frederik Träuble, Peter Gehler, Chris Russell, Matthias Bethge, Bernhard Schölkopf, Francesco Locatello, and Wieland Brendel. Visual representation learning does not generalize strongly within the same domain. *arXiv preprint arXiv:2107.08221*, 2021. [A.1.2](#)
- [71] John Schulman. Trust region policy optimization. *arXiv preprint arXiv:1502.05477*, 2015. [2.1](#)
- [72] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. [2.1](#), [2.3](#), [2.3.1](#)
- [73] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. [2.4.1](#), [3.1](#), [3.4.3](#), [3.6](#)
- [74] Yang Song and Prafulla Dhariwal. Improved techniques for training consistency models. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=WNzy9bRDvG>. [3.1](#)
- [75] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020. [2.4.1](#), [3.3.4](#)
- [76] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. *arXiv preprint arXiv:2303.01469*, 2023. [1](#), [3.1](#), [3.2.3](#), [3.5.3](#), [3.6](#), [3.6](#)
- [77] Yeda Song, Dongwook Lee, and Gunhee Kim. Compositional conservatism: A transductive approach in offline reinforcement learning. *arXiv preprint arXiv:2404.04682*, 2024. [2.1](#)
- [78] Jan Stanczuk, Georgios Batzolis, Teo Deveney, and Carola-Bibiane Schönlieb. Your diffusion model secretly knows the dimension of the data manifold. *arXiv*

- preprint arXiv:2212.12611*, 2022. [A.2](#)
- [79] Guanquan Wang, Takuya Hiraoka, and Yoshimasa Tsuruoka. Planning with consistency models for model-based offline reinforcement learning. In *Deployable RL: From Research to Practice@ Reinforcement Learning Conference 2024*. [3.6](#)
  - [80] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022. [2.5.2](#), [3.3.1](#), [3.4.1](#), [3.6](#)
  - [81] Zhendong Wang, Zhaoshuo Li, Ajay Mandlekar, Zhenjia Xu, Jiaojiao Fan, Yashraj Narang, Linxi Fan, Yuke Zhu, Yogesh Balaji, Mingyuan Zhou, et al. One-step diffusion policy: Fast visuomotor policies via diffusion distillation. *arXiv preprint arXiv:2410.21257*, 2024. [3.1](#), [3.6](#)
  - [82] Thaddäus Wiedemer, Prasanna Mayilvahanan, Matthias Bethge, and Wieland Brendel. Compositional generalization from first principles. *Advances in Neural Information Processing Systems*, 36, 2024. [2.2.1](#), [2.2.2](#), [A.1.2](#)
  - [83] Chao Yu, Akash Velu, Eugene Vinitzky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35:24611–24624, 2022. [2.6.3](#)
  - [84] 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. [A.1](#), [A.1.1](#)
  - [85] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33:14129–14142, 2020. [3.4.1](#)
  - [86] Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018. [A.1.1](#)
  - [87] Linfeng Zhao, Lingzhi Kong, Robin Walters, and Lawson LS Wong. Toward compositional generalization in object-oriented world modeling. In *International Conference on Machine Learning*, pages 26841–26864. PMLR, 2022. [2.1](#)