# Goal-Directed Exploration and Skill Reuse

*Vitchyr Pong*

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-176
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-176.html

August 10, 2021

Goal-Directed Exploration and Skill Reuse

by

Vitchyr He Pong

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Sergey Levine, Chair
Professor Pieter Abbeel
Professor Trevor Darrell
Professor Alison Gopnik

Summer 2021

Goal-Directed Exploration and Skill Reuse

Abstract

Goal-Directed Exploration and Skill Reuse

by

Vitchyr He Pong

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Sergey Levine, Chair

Reinforcement learning is a powerful paradigm for training agents to acquire complex behaviors, but it assumes that an external reward is provided by the environment. In practice, this task supervision is often hand-crafted by a user, a process that is time-consuming to repeat for every possible task and that makes manual engineering a primary bottleneck for behavior acquisition. This thesis describes how agents can acquire and reuse goal-directed behaviors in a completely self-supervised manner. It discusses challenges that arise when scaling up these methods to complex environments: How can an agent set goals for itself when it does not even know the set of possible states to explore? How does an agent autonomously reward itself for reaching a goal? How can an agent reuse this goal-directed behavior to decompose a new task into easier goal-reaching tasks? This thesis presents methods that I have developed to address these problems and share results that apply the methods to image-based, robot environments.

To my parents and sister.

# Contents

**Contents** ii

**List of Figures** v

**List of Tables** xv

**1 Introduction** 1

**2 Goal-Conditioned Reinforcement Learning** 5
- 2.1 Background . . . 5
  - 2.1.1 Goal-conditioned reinforcement learning. . . . 7
  - 2.1.2 Variational Autoencoders. . . . 8
- 2.2 Related Work . . . 9
- 2.3 Goal-Reaching as Probabilistic Inference . . . 13
  - 2.3.1 Outcome-Driven Reinforcement Learning . . . 14
  - 2.3.2 Warm-up: Achieving a Desired Outcome at a Fixed Time Step . . . 15
  - 2.3.3 Outcome-Driven Reinforcement Learning as Variational Inference . . . 17
  - 2.3.4 Outcome-Driven Reinforcement Learning . . . 19
  - 2.3.5 Empirical Evaluation . . . 22
- 2.4 Conclusion . . . 24

**3 Generating Goals for Autonomous Practice** 25
- 3.1 Reinforcement Learning with Imagined Goals . . . 25
  - 3.1.1 Goal-Conditioned Policies with Unsupervised Representation Learning 26
  - 3.1.2 Experiments . . . 30
- 3.2 Skew-Fit: Setting the Right Goals . . . 35
  - 3.2.1 Problem Formulation . . . 37
  - 3.2.2 Skew-Fit: Learning a Maximum Entropy Goal Distribution . . . 38
  - 3.2.3 Training Goal-Conditioned Policies with Skew-Fit . . . 42
  - 3.2.4 Experiments . . . 43
- 3.3 Conclusion . . . 49

**4 Reusing Goal-Directed Behavior** 51

4.1 Temporal Difference Models . . . 52
4.1.1 Model-based RL and optimal control. . . . 52
4.1.2 Temporal Difference Model Learning . . . 53
4.1.3 Training and Using Temporal Difference Models . . . 55
4.1.4 Experiments . . . 57
4.2 Planning with Images . . . 60
4.2.1 Planning with Goal-Conditioned Policies . . . 60
4.2.2 Experiments . . . 64
4.3 Conclusion . . . 68

**5 Extensions to Non-Goal-Reaching Tasks** **70**
5.1 Distribution-Conditioned Reinforcement Learning . . . 70
5.1.1 Contextual Markov Decision Processes . . . 71
5.1.2 Distribution-Conditioned Reinforcement Learning . . . 72
5.1.3 Learning Goal Distributions and Policies . . . 74
5.1.4 Experiments . . . 77
5.2 Contextual and Meta-Reinforcement Learning . . . 80
5.2.1 Related Works in Meta-Reinforcement Learning . . . 81
5.2.2 Preliminaries . . . 83
5.2.3 The Problem with Naïve Offline Meta-Reinforcement Learning . . . 84
5.2.4 Semi-Supervised Meta Actor-Critic . . . 86
5.2.5 Experiments . . . 89
5.3 Conclusion . . . 93

**6 Discussion and Future Work** **95**

**Bibliography** **97**

**A Contributions** **118**

**B Chapter 2 Appendix** **119**
B.1 Proofs & Derivations . . . 119
B.1.1 Derivation of Variational Objectives . . . 119
B.1.2 Derivation of Recursive Variational Objective . . . 122
B.1.3 Derivation of Optimal Variational Posterior over $T$ . . . 131
B.1.4 Lemmas . . . 134
B.2 Proof of Outcome-Driven Policy Iteration Theorem . . . 139
B.3 Additional Experiments . . . 143
B.4 Experimental Details . . . 148
B.4.1 Environment . . . 148
B.4.2 Algorithm . . . 148
B.4.3 Implementation . . . 149

**C Chapter 3 Appendix** **152**
C.1 Section 3.1 Appendix . . . 152
C.1.1 Complete Ablative Results . . . 152
C.1.2 Hyperparameters . . . 155
C.1.3 Environment Details . . . 155
C.2 Section 3.2 Appendix . . . 156
C.2.1 Proofs . . . 156
C.2.2 Additional Experiments . . . 160
C.2.3 Implementation Details . . . 164
C.2.4 Environment Details . . . 168
C.2.5 Goal-Conditioned Reinforcement Learning Minimizes $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$ . . . 173

**D Chapter 4 Appendix** **174**
D.1 Section 4.1 Appendix . . . 174
D.1.1 Experiment Details . . . 174
D.2 Section 4.2 Appendix . . . 178
D.2.1 Additional Experiments . . . 178
D.2.2 Environment Details . . . 180
D.2.3 Implementation Details . . . 182

**E Chapter 5 Appendix** **185**
E.1 Section 5.1 Appendix . . . 185
E.1.1 Additional Results . . . 185
E.1.2 Environments . . . 185
E.1.3 Experimental Details . . . 188
E.1.4 Implementation Details . . . 190
E.2 Section 5.2 Appendix . . . 193
E.2.1 Additional Experimental Results . . . 193
E.2.2 Experimental Details . . . 193

# List of Figures

2.1 Illustration of the shaping effect of the reward function derived from the goal-directed variational inference objective. (Left) A 2-dimensional grid world with a desired outcome marked by a star. (Middle left) The corresponding sparse reward function provides little shaping. (Middle right) The reward function derived from our variational inference formulation at initialization. (Right) The derived reward function after training. We see that the derived reward learns to provide a dense reward signal everywhere in the state space. . . . . 14

2.2 A probabilistic graphical model of a state–action trajectory with observed random variables $\mathbf{s}_0$ and $\mathbf{s}_{t^\star}$.

. . . . . 15

2.3 From left to right, we evaluate on: a 2D environment in which an agent must move around a box, a locomotion task in which a quadruped robot must match a location and pose (yellow), and four manipulation tasks in which the robot must push objects, rotate faucet valve, or open a window. . . . . 21

2.4 Learning curves showing final distance vs environment steps across all six environments. We see that only ODAC consistently performs well on all six tasks. Prior methods struggle to learn, especially in the absence of uniform goal sampling. See text for details. . . . . 22

3.1 We train a VAE using data generated by our exploration policy (left). We use the VAE for multiple purposes during training time (middle): to sample goals to train the policy, to embed the observations into a latent space, and to compute distances in the latent space. During test time (right), we embed a specified goal observation $o_g$ into a goal latent $\mathbf{z}_g$ as input to the policy. Videos of our method can be found at `sites.google.com/site/visualrlwithimaginedgoals` . . . . 27

3.2 (Left) The simulated pusher, door opening, and pick-and-place environments are pictured. (Right) Test rollouts from our learned policy on the three pushing environments. Each row is one rollout. The right two columns show a goal image $\mathbf{g}$ and its VAE reconstruction $\hat{\mathbf{g}}$. The images to their left show frames from a trajectory to reach the given goal. . . . . 30

3.3 Simulation results, final distance to goal vs simulation steps[1]. RIG (red) consistently outperforms the baselines, except for the oracle which uses ground truth object state for observations and rewards. On the hardest tasks, only our method and the oracle discover viable solutions. . . . . 32
3.4 Reward type ablation results. RIG (red), which uses latent Euclidean distance, outperforms the other methods. . . . . 32
3.5 Training curve for relabeling ablation. We see that the RIG-style relabeling that relabels with both future and self-generated goals performs the best. . . . . 33
3.6 Training curve for learning with varying number of objects. We see that RIG makes progress on this task, which does not length itself to a fix-length state representation. . . . . 33
3.7 (Left) Our method compared to the HER baseline and oracle on a real-world visual reaching task. (Middle) Our robot setup is pictured. (Right) Test rollouts of our learned policy. . . . . 34
3.8 (Left) The learning curve for real-world pushing. (Middle) Our robot pushing setup is pictured, with frames from test rollouts of our learned policy. (Right) Our method compared to the HER baseline on the real-world visual pushing task. We evaluated the performance of each method by manually measuring the distance between the goal position of the puck and final position of the puck for 15 test rollouts, reporting mean and standard deviation. . . . . 34
3.9 Left: Robot learning to open a door with Skew-Fit, without any task reward. Right: Samples from a goal distribution when using (a) uniform and (b) Skew-Fit sampling. When used as goals, the diverse samples from Skew-Fit encourage the robot to practice opening the door more frequently. . . . . 36
3.10 Our method, Skew-Fit, samples goals for goal-conditioned RL. We sample states from our replay buffer, and give more weight to rare states. We then train a generative model $q^G_{\phi_{t+1}}$ with the weighted samples. By sampling new states with goals proposed from this new generative model, we obtain a higher entropy state distribution in the next iteration. . . . . 39
3.11 Illustrative example of Skew-Fit on a 2D navigation task. (Left) Visited state plot for Skew-Fit with $\alpha = -1$ and uniform sampling, which corresponds to $\alpha = 0$. (Right) The entropy of the goal distribution per iteration, mean and standard deviation for 9 seeds. Entropy is calculated via discretization onto an 11x11 grid. Skew-Fit steadily increases the state entropy, reaching full coverage over the state space. . . . . 43
3.12 (Left) Ant navigation environment. (Right) Evaluation on reaching target XY position. We show the mean and standard deviation of 6 seeds. Skew-Fit significantly outperforms prior methods on this exploration task. . . . . 44
3.13 We evaluate on these continuous control tasks, from left to right: *Visual Door*, a door opening task; *Visual Pickup*, a picking task; *Visual Pusher*, a pushing task; and *Real World Visual Door*, a real world door opening task. All tasks are solved from images and without any task-specific reward. See Appendix C.2.4 for details. 45

3.14 Learning curves for simulated continuous control tasks. Lower is better. We show the mean and standard deviation of 6 seeds and smooth temporally across 50 epochs within each seed. Skew-Fit consistently outperforms RIG and various prior methods. See text for description of each method. . . . . 47
3.15 Cumulative total pickups during exploration for each method. Prior methods fail to pay attention to the object: the rate of pickups hardly increases past the first 100 thousand timesteps. In contrast, after seeing the object picked up a few times, Skew-Fit practices picking up the object more often by sampling the appropriate exploration goals. . . . . 48
3.16 (Top) Learning curve for Real World Visual Door. Skew-Fit results in considerable sample efficiency gains over RIG on this real-world task. (Bottom) Each row shows the Skew-Fit policy starting from state $\mathbf{s}_1$ and reaching state $\mathbf{s}_{100}$ while pursuing goal $\mathbf{G}$. Despite being trained from only images without any user-provided goals during training, the Skew-Fit policy achieves the goal image provided at test-time, successfully opening the door. . . . . 49

4.1 The tasks in our experiments: (a) reaching target locations, (b) pushing a puck to a random target, (c) training the cheetah to run at target velocities, (d) training an ant to run to a target position or a target position and velocity, and (e) reaching target locations (real-world Sawyer robot). . . . . 58
4.2 The comparison of TDM with model-free (DDPG, both with sparse and dense rewards), model-based, and goal-conditioned value functions (HER with sparse rewards) methods on various tasks. All plots show the final distance to the goal versus 1000 environment steps (not rollouts). The bold line shows the mean across 3 random seeds, and the shaded region show one standard deviation. Our method, which uses model-free learning, is generally more sample-efficient than model-free alternatives including DDPG and HER and improves upon the best model-based performance. . . . . 58
4.3 Ablation experiments for (a) scalar vs. vectorized TDMs on 7-DoF simulated reacher task and (b) different $t_{max}$ on pusher task. The vectorized variant performs substantially better, while the horizon effectively interpolates between model-based and model-free learning. . . . . 59
4.4 Summary of Latent Embeddings for Abstracted Planning (LEAP). (1) The planner is given a goal state. (2) The planner plans intermediate subgoals in a low-dimensional latent space. By planning in this latent space, the subgoals correspond to valid state observations. (3) The goal-conditioned policy then tries to reach the first subgoal. After $t_1$ time steps, the policy replans and repeats steps 2 and 3. 61
4.5 Optimizing directly over the image manifold (b) is challenging, as it is generally unknown and resides in a high-dimensional space. We optimize over a latent state (a) and use our decoder to generate images. So long as the latent states have high likelihood under the prior (green), they will correspond to realistic images, while latent states with low likelihood (red) will not. . . . . 62

4.6 Comparisons on two vision-based domains that evaluate temporally extended control, with illustrations of the tasks. In 2D Navigation (left), the goal is to navigate around a U-shaped wall to reach the goal. In the Push and Reach manipulation task (right), a robot must first push a puck to a target location (blue star), which may require moving the hand away from the goal hand location, and then move the hand to another location (red star). Curves are averaged over multiple seeds and shaded regions represent one standard deviation. Our method, shown in red, outperforms prior methods on both tasks. On the Push and Reach task, prior methods typically get the hand close to the right location, but perform much worse at moving the puck, indicating an overly greedy strategy, while our approach succeeds at both. . . . . 66

4.7 (Left) Visualization of subgoals reconstructed from the VAE (bottom row), and the actual images seen when reaching those subgoals (top row). Given an initial state $s_0$ and a goal image $\mathbf{g}$, the planner chooses meaningful subgoals: at $\mathbf{g}_{t_1}$, it moves towards the puck, at $\mathbf{g}_{t_2}$ it begins pushing the puck, and at $\mathbf{g}_{t_3}$ it completes the pushing motion before moving to the goal hand position at $\mathbf{g}$. (Middle) The top row shows the image subgoals superimposed on one another. The blue circle is the starting position, the green circle is the target position, and the intermediate circles show the progression of subgoals (bright red is $\mathbf{g}_{t_1}$, brown is $\mathbf{g}_{t_3}$). The colored circles show the subgoals in the latent space (bottom row) for the two most active VAE latent dimensions, as well as samples from the VAE aggregate posterior [144]. (Right) Heatmap of the value function $V(\mathbf{s}, \mathbf{g}, t)$, with each column showing a different time horizon $t$ for a fixed state $\mathbf{s}$. Warmer colors show higher value. Each image indicates the value function for all possible goals $\mathbf{g}$. As the time horizon decreases, the value function recognizes that it can only reach nearby goals. . . . . 67

4.8 In the Ant Navigation task, the ant must move around the long wall, which will incur large negative rewards during the trajectory, but will result in an optimal final state. We illustrate the task, with the purple ant showing the starting state and the green ant showing the goal. We use 3 subgoals here for illustration. Our method (shown in red in the plot) is the only method that successfully navigates the ant to the goal. . . . . 68

4.9 (Left) Ablative studies on 2D Navigation. We keep all components of LEAP the same but replace optimizing over the latent space with optimizing over the image space (-latent). We separately train the RL methods from scratch rather than reusing the VAE mean encoder (-shared), and also test both ablations together (-latent, shared). We see that sharing the encoder weights with the RL policy results in faster learning, and that optimizing over the latent space is critical for success of the method. (Right) Visualization of the subgoals generated when optimizing over the latent space and decoding the image (top) and when optimizing over the images directly (bottom). The goals generated when planning in image space are not meaningful, which explains the poor performance of "-latent" shown in (Left). . . . . 69

5.1 We infer the distribution parameters $\omega$ from data and pass it to a DisCo policy. Distribution-conditioned RL can express a broad range of tasks, from defining relationships between different state components (top) to more arbitrary behavior (bottom). . . . . 71

5.2 A robot must arrange objects into a configuration $\mathbf{s}_f$ that changes from episode to episode. This task consists of multiple sub-tasks, such as first moving the red object to the correct location. Given a final state $\mathbf{s}_f$, there exists a distribution of intermediate states $\mathbf{s}$ in which the first sub-task is completed. We use pairs of states $\mathbf{s}, \mathbf{s}_f$ to learn a conditional distribution that defines the first sub-task given the final task, $p(\mathbf{s} \mid \mathbf{s}_f)$. . . . . 75

5.3 Illustrations of the experimental domains, in which a policy must (left top) use the blue cursor to move objects to different locations, (left bottom) control a Sawyer arm to move cubes into and out of a box, and (right) attach shelves to a pole using a cursor. . . . . 78

5.4 (Lower is better.) Learning curve showing normalized distance versus environment steps of various method. DisCo RL uses a (left) Gaussian model, (middle) Gaussian mixture model, or (right) latent-variant model on their respective tasks. DisCo RL with a learned goal distribution consistently outperforms VICE and obtains a final performance similar to using oracle rewards. . . . . 79

5.5 (Higher is better.) Learning curves showing the number of cumulative tasks completed versus environment steps for the Sawyer (left), IKEA (middle), and Flat World (right) tasks. We see that DisCo RL significantly outperforms HER and VICE, and that relabeling the mean and covariance is important. . . . . 80

5.6 (left) In offline meta-RL, an agent uses offline data from multiple tasks $T_1, T_2, \ldots$, each with reward labels that must only be provided once. (middle) In online meta-RL, new reward supervision must be provided with every environment interaction. (right) In semi-supervised meta-RL, an agent uses an offline dataset collected once to learn to generate its own reward labels for new, online interactions. Similar to offline meta-RL, reward labels must only be provided once for the offline training, and unlike online meta-RL, the additional environment interactions do not require external reward supervision. . . . . 82

5.7 **Left:** The distribution of the KL-divergence between the posterior $q_\phi(\mathbf{z} \mid \mathbf{h})$ and a prior $p(\mathbf{z})$ over the course of training, when conditioned on data from the offline dataset (blue) or learned policy (orange), as measured by $D_{\mathrm{KL}}(q_\phi(\mathbf{z} \mid \mathbf{h}) || p(\mathbf{z}))$. We see that data from the learned policy results in posteriors that are substantially farther from the prior, suggesting a significant difference in distribution over $\mathbf{z}$. **Right:** The performance of the policy post-adaptation when conditioned on data from the offline dataset (i.e., $\mathbf{z} \sim p(\mathbf{z} \mid \pi_\beta)$) and data generated by the learned policy (i.e., $\mathbf{z} \sim p(\mathbf{z} \mid \pi)$). During the offline training phase, we see that although the meta-RL policy learns when conditioned on $\mathbf{z}$ generated by the offline data, the performance does not increase when $\mathbf{z}$ is generated using the online data. Since the same policy is evaluated, the change in $\mathbf{z}$-distribution is likely the cause for the drop in performance. In contrast, during the self-supervised training phase, the performance of the two is quite similar. . . . . 85

5.8 (Left) In the offline phase, we sample a history $\mathbf{h}'$ to compute the posterior $q_\phi(\mathbf{z} \mid \mathbf{h}')$. We then use a sample from this encoder and another history $\mathbf{h}$ to train the networks. In purple, we update the encoder $q_\phi$ with both reconstruction and KL loss. (Right) During the self-supervised phase, we explore by sampling $\mathbf{z} \sim p(z)$ and conditioning our policy on these observations. We label rewards using our learned reward decoder, and append the resulting data to the training data. The training procedure is equivalent to the offline phase, except that we do not train the reward decoder or encoder since no additional ground-truth rewards are observed. . . . . 86

5.9 Examples of our evaluation domains, each of which has a set of meta-train tasks (examples shown in blue) and held out test tasks (orange). The domains include (left) a half cheetah tasked with running at different speeds, (middle) a quadruped ant locomoting to different points on a circle, and (right) a simulated Sawyer arm performing various manipulation tasks. . . . . 90

5.10 Comparison on self-supervised meta-learning against baseline methods. We report the final return of meta-test adaptation on unseen test tasks, with varying amounts of online meta-training following offline meta-training. Our method SMAC, shown in red, consistently trains to a reasonable performance from offline meta-RL (shown at step 0) and then steadily improves with online self-supervised experience. The offline meta-RL methods, MACAW [148] and BOReL at best match the offline performance of SMAC but have no mechanism to improve via self-supervision. We also compare to SMAC (SAC ablation) which uses SAC instead of AWAC as the underlying RL algorithm. This ablation struggles to pretrain a value function offline, and so struggles to improve on more difficult tasks. . . . . 91
5.11 We visualize the visited XY-coordinates of the learned policy on the Ant Direction task. **Left:** Trajectories from the post-adaptation policy conditioned on $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{h})$ when $\mathbf{h}$ is sampled from the offline dataset (blue) or the learned exploration policy (orange) immediately after offline training. When conditioned on offline data, the policy correctly moves in many different directions. However, when conditioned on data from the learned exploration policy, the post-adaptation policy only moves up and to the left, suggesting that the post-adaptation policy is sensitive to data distribution used to collect $\mathbf{h}$. **Right:** After the self-supervised phase, we see that the post-adaptation policy learns to move in many different directions regardless of the data source. These visualization demonstrate that the self-supervised phase mitigates the distribution shift between conditioning on offline and online data. . . . . 92

B.1 Ablation results across all six environments. We see that using our derived $q_T$ equation is important for best performance across all six tasks and that ODAC is not sensitive to the quality of dynamics model. . . . . 145
B.2 Comparison of different methods when the desired outcome $\mathbf{g}$ is sampled uniformly from the set of possible outcomes during exploration. In this easier setting, we see that the UVD performance is similar to ODAC. These results suggest that UVD depends more heavily on sampling outcomes from the set of desired outcomes than ODAC. . . . . 145
B.3 The inferred $q_{\Delta_{t+1}}(\Delta_{t+1} = 0)$ versus time during an example trajectory in the Ant environment. As the ant robot falls over, $q_{\Delta_{t+1}}(\Delta_{t+1} = 0)$ drops in value. We see that the optimal posterior $q^\star_{\Delta_{t+1}}(\Delta_{t+1} = 0)$ given in Proposition 2 automatically assigns a high likelihood of terminating when this irrecoverable state is first reached, effectively acting as a dynamic discount factor. . . . . 146

B.4 We visualize the rewards over the course of training for the Box 2D environment. The darkest color corresponds to a reward of $-100$ and brightest to 0. To visualize the reward, we discretize the continuous state space and evaluate $r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta)$ for $\mathbf{a}_t = \vec{0}$ at different states. As shown in Figure B.4c, the desired outcome $\mathbf{g}$ is near the bottom right. After 4-8 thousand environment steps, the reward is more flat near $\mathbf{g}$, and only provides a reward gradient far from $\mathbf{g}$. After 20 thousand environment steps, the reward gradient is much larger again near the end, and the penalty for being in the top left corner has changed from $-1.6$ to $-107$. . . 147

C.1 Relabeling ablation simulated results, showing final distance to goal vs environment steps. RIG (red), which uses a mixture of VAE and future, consistently matches or outperforms the other methods. . . . . 152

C.2 Reward type ablation simulated results, showing final distance to goal vs environment steps. RIG (red), which uses latent distance for the reward, consistently matches or outperforms the other reward types. . . . . 153

C.3 Online vs offline VAE training ablation simulated results, showing final distance to goal vs environment steps. Given no pre-training phase, training the VAE online (red), outperforms no training of the VAE, and also performs well. . . . . 154

C.4 Comparison between our relabeling strategy and HER. Each column shows a different task from the OpenAI Fetch robotics suite. The top row uses 64 gradient updates per training cycle and the bottom row uses 256 updates per cycle. Our relabeling strategy is significantly better for both sparse and dense rewards, and for higher number of updates per cycle. . . . . 154

C.5 We compare using SAC [84] and TD3 [71] as the underlying RL algorithm on Visual Door, Visual Pusher and Visual Pickup. We see that Skew-Fit works consistently well with both SAC and TD3, demonstrating that Skew-Fit may be used with various RL algorithms. For the experiments presented in subsection 3.2.4, we used SAC. . . . . 161

C.6 We sweep different values of $\alpha$ on Visual Door, Visual Pusher and Visual Pickup. Skew-Fit helps the final performance on the Visual Door task, and outperforms No Skew-Fit ($\alpha = 0$) as seen in the zoomed in version of the plot. In the more challenging Visual Pusher task, we see that Skew-Fit consistently helps and halves the final distance. Similarly, we observe that Skew-Fit consistently outperforms No Skew-fit on Visual Pickup. Note that alpha=-1 is not always the optimal setting for each environment, but outperforms $\alpha = 0$ in each case in terms of final performance. . . . . 162

C.7 Gradient variance averaged across parameters in last epoch of training VAEs. Values of $\alpha$ less than $-1$ are numerically unstable for importance sampling (IS), but not for Skew-Fit. . . . . 163

C.8 Proposed goals from the VAE for RIG and with Skew-Fit on the *Visual Pickup*, *Visual Pusher*, and *Visual Door* environments. Standard RIG produces goals where the door is closed and the object and puck is in the same position, while RIG + Skew-Fit proposes goals with varied puck positions, occasional object goals in the air, and both open and closed door angles. . . . . 165
C.9 Proposed goals from the VAE for RIG (left) and with RIG + Skew-Fit (right) on the *Real World Visual Door* environment. Standard RIG produces goals where the door is closed while RIG + Skew-Fit proposes goals with both open and closed door angles. . . . . 166
C.10 Example reached goals by Skew-Fit and RIG. The first column of each environment section specifies the target goal while the second and third columns show reached goals by Skew-Fit and RIG. Both methods learn how to reach goals close to the initial position, but only Skew-Fit learns to reach the more difficult goals. . . . 167

D.1 TDMs with different number of updates per step $I$ on ant target position task. The maximum distance was set to 5 rather than 6 for this experiment, so the numbers should be lower than the ones reported in the paper. . . . . 175
D.2 We compare using the $\ell_\infty$-norm to the $\ell_1$-norm. We see that the $\ell_\infty$-norm outperforms the $\ell_1$-norm. . . . . 179
D.3 We compare CEM to different optimizers L-BFGS, Adam, RMSProp, and gradient descent (SGD) that have had their learning rates tuned. (Left) The optimizer loss, where CEM outperforms the other methods. (Right) The performance of the policy after using the plan chosen by each optimizer. We see that the lower optimizer loss of CEM corresponds to a better performance. 179
D.4 Examining the effect of the weight $\lambda$ in Equation 4.10. We note the final RL performance (left), log-likelihood under the VAE prior (middle), and V values (right). As we increase $\lambda$, the log-likelihood values increase while the V values decrease. For 2D navigation (top), we note the optimal value to be $\lambda = 0.01$ and for Push and Reach (bottom) any range of values between 0.0001 and 0.01. . . . . 180
D.5 Complete 2D Navigation Results . . . . . 181
D.6 Complete Push and Reach Results . . . . . 182
D.7 Complete Ant Navigation Results . . . . . 182

E.1 (a) The distance between the final end-effector position and the position specified in the goal state $\mathbf{s}_g$. (b) The success rate of the methods on the same task (with results copied from Figure 5.5 for convenience). We see that goal-conditioned RL focuses primarily on moving the end effector to the correct position, while DisCo RL ignores this task-irrelevant dimension and successfully completes the task. . 186
E.2 Example images with the hand in a fixed position used to obtain a goal distribution. 187

E.3 We duplicate Figure 5.11 but include the exploration trajectories (green) and example trajectories from the offline dataset (red). We see that the exploration policy both before and after self-supervised training primarily moves up and to the left, whereas the offline data moves in all direction. Before the self-supervised phase, we see that conditioning the encoder on online data (orange) rather than offline data (blue) results in very different policies, with the online data resulting in the post-adaptation policy only moving up and to the left. However, the self-supervised phase of SMAC mitigates the impact of this distribution shift and results in qualitatively similar post-adaptation trajectories, despite the large difference between the exploration trajectories and offline dataset trajectories. . 194

# List of Tables

2.1 Ablation results, showing mean final normalized distance (×100) at the end of training across 4 seeds. Best mean is in bold and standard error in parentheses. ODAC is not sensitive to the dynamics models $\hat{p}_d$ but benefits from the dynamic $q_T$ variant. . . . . 23

B.1 Normalized final distances (lower is better) across four random seeds, multiplied by a factor of 100. . . . . 146
B.2 Environment specific hyper-parameters. . . . . 151
B.3 General hyper-parameters used for all experiments. . . . . 151

C.1 Hyper-parameters used for all experiments. . . . . 155
C.2 Despite training on a unbalanced Visual Door dataset (see Figure 7 of paper), the negative log-likelihood (NLL) of Skew-Fit evaluated on a uniform dataset matches that of a VAE trained on a uniform dataset. . . . . 163
C.3 General hyper-parameters used for all *visual* experiments. . . . . 169
C.4 Environment specific hyper-parameters for the *visual* experiments . . . . . 170
C.5 Hyper-parameters used for the *ant* experiment. . . . . 170

D.1 TD3 [71] hyperparameters. . . . . 183

E.1 Environment specific final goal distributions. . . . . 189
E.2 Environment specific hyper-parameters. . . . . 190
E.3 General hyper-parameters used for all experiments. . . . . 190
E.4 SMAC Hyperparameters for Self-Supervised Phase . . . . . 196
E.5 Environment Specific SMAC Hyperparameters . . . . . 196

## Acknowledgments

I thank my advisor Sergey Levine, for his patience and guidance these past few years. I could have not done this research and become the researcher I am today without you.

I thank my many fellow PhD collaborators for helping me conduct the research presented in this thesis, including Shixiang Gu, Aurick Zhou, Tuomas Haarnoja, Abhishek Gupta, Gregory Kahn, Tim G. J. Rudner, Laura Smith, I am especially grateful to have worked with Ashvin Nair, whose faith and diligence inspired me as we worked together through various brainstorming sessions, code refactors, and research projects. I thank the many undergraduate and masters students that I had the privilege to work with, including Shikhar Bahl, Alexander Khazatsky, Catherine Huang, Kevin Li, Justin Yu, and Ashwin Reddy. So much of this work could not have been possible without all of you. To Murtaza Dalal, Steven Lin, and Soroush Nasiriany in particular: thank you for being wonderful collaborators and teaching me so much.

I am honored to have received the advice of so many wonder faculty members, including Pieter Abbeel, Trevor Darrell, Yarin Gal, and Alison Gopnik. I thank Tom Schaul and the DeepMind Reinforcement Learning team for helping me grow as a researcher and thinker. I thank Roberto Calandra, Chelsea Finn, Dinesh Jayaraman, Rowan McAllister, and Glen Berseth for the advice they shared, which took me my entire PhD to understand.

I thank Alyosha Molnar, Carl Poitras, and David Albonesi for sparking my interest in research and encouraging me to pursue a PhD. I thank Hadas Kress-Gazit for providing me with an early research opportunity. I thank Ross Knepper for introducing me to robotics and providing me much-needed mentorship early in my research career.

I am so grateful for all the friends I made in graduate school, including Parsa Mahmoudieh, John D. Co-Reyes, Marvin Zhang, Justin Fu, Avi Singh, Kelvin Xu, Deirdre Quillen, Erin Grant, Frederik Ebert, Michael B. Chang, Michael Janner, Dibya Ghosh, Pim de Haan, Carlos Florensa, Anusha Nagabandi, Kate Rakelly, Alex Lee, Adam Stooke, Coline Devin, Natasha Jaques, Sid Reddy, Pulkit Agrawal, Tuomas Haarnoja, Jasmine Collins, Xinyang Geng, Efe Aras, Abhishek Gupta, and Ashvin Nair. Thank you for the memories of eating at Euclid St., dumpling and board game nights, late night or even mid-day philosophising, whiteboard sessions that were much too loud for the other SDH occupants, San Francisco escapades, commiserating over the PhD experience, epic post-conference trips, and many more fond memories that I will cherish.

I thank Matt McPhail, Jeremy Warner, Nehal Molasarria, and Elli Kang for being awesome roommates, listening to me whenever I needed an ear, and kicking my butt at Dominion and Overcooked. Suma Anand, Andreea Bobu, and Vickie Ye: thank you for the Harry Potter binge, balcony brunches, Jackbox sessions, Insanity and Blogilates workouts, four-person dance parties, and countless dinners that made the past couple of years a blast.

I thank my amazing family, Fang He, Muy Long Pong, and Vidie Pong, who were always there to remind me of who I was when I got lost. Vickie Ye, thank you for your generous love and support that helped me grow and filled me with joy. Last, and most of all, I thank God for His unwavering love and for blessing me with all these wonderful people in my life.

# Chapter 1

# Introduction

General-purpose robots that exhibit a broad range of capabilities in unstructured environments would greatly advance the utility of robotics. However, robot applications have traditionally been limited to domains, such as manufacturing or chemical engineering, in which the environment is highly predictable by design or for which we have accurate models. In such environments, one can manually design or synthesize a robot controller using prior knowledge of the environment dynamics, sensors, and workspace configuration [5, 151]. However, such a degree of predictability is often missing in many potential robot applications. In applications such as construction, household robotics, and autonomous vehicles, robots must operate "in the wild" with varying workspace configurations and desired behaviors.

Developing general-purpose robots for broader applications will require overcoming a number of challenges. One limitation with manually designed controllers is that the design process must be repeated for each new task. In unstructured environments, one cannot practically anticipate and manually program a controller for every possible situation or task, and so a promising ingredient for developing general-purpose robots is to enable robots to acquire new manipulation and locomotion capabilities completely autonomously, without requiring user effort. Even if skill acquisition was automated, another challenge is that there are exponentially many potential scenarios and tasks for which we may want to deploy a robot. To handle the diversity of tasks we may care about, we need to develop methods that allow robots to reuse their knowledge for solving old tasks to quickly solve new tasks, rather than relying on robots to acquire each skill independently. Lastly, to deploy robots in complex, unstructured environments where the number of sensors may be limited and the configuration of the workspace may not be known beforehand, we need to develop methods that can operate directly from raw sensory observations, such as images.

A promising approach to handle unstructured environments is a data-driven one that combines techniques from reinforcement learning and deep learning. Reinforcement learning (RL) agents can learn complex behaviors such as stratospheric balloon control [14] or data center cooling [127, 137] by simply providing a reward. Similar to how a person can train a dog by providing treats rather than by commanding its exact muscle movements, RL enables a user to train an agent by only evaluating behaviors rather than by designing

controllers. In parallel, machine learning as a field has seen great advances in optimizing neural networks, which consist of differentiable operations stacked together to produce deep, expressive function approximators. These recent advances in "deep learning" have produced great strides in modeling high-dimensional data, such as images [19, 112, 46] and natural language text [213, 45, 21], while providing a shared substrate for optimizing decision-making objectives. The confluence of deep learning and reinforcement learning has produced impressive results, such as achieving professional or superhuman performance in video games [15, 216] and board games [193]. Although deep reinforcement learning methods do not require extensive prior knowledge and can learn directly from high-dimensional data, the standard deep reinforcement learning paradigm is unlikely to address the remaining challenges with training general-purpose robots.

A major shortcoming with deep reinforcement learning is that it traditionally trains agents to maximize a single, fixed reward. As a result, a robot trained with standard RL is specialized to one specific task and cannot reuse its learned skills to solve new tasks. Moreover, the skill acquisition loop is by no means automated: traditional reinforcement learning still relies on an external reward that must be provided for every action that an agent takes. This reward must be manually provided by a user or be provided indirectly by a programmer that writes code to automatically assign rewards. In other words, although RL requires a user to "only" provide rewards for each new task, it still requires significant effort from a user for each task that a user wants a robot to accomplish.

**Beyond standard reinforcement learning.** In order to enable agents to autonomously acquire reusable skills, we need to expand the traditional reinforcement learning paradigm to enable an agent to set tasks for themselves and reward themselves, while also providing a mechanism to reuse the learned policies to solve new tasks. To this end, this thesis studies how an agent can set goals for itself and reuse goal-directed capabilities in the context of *goal-conditioned* reinforcement learning. In goal-conditioned RL, the objective of an agent is to reach a goal that is provided to the agent. Unlike standard reinforcement learning which would train a separate policy for each goal, goal-conditioned RL trains one policy to reach many goals by conditioning the policy directly on the goal.

Goal-conditioned reinforcement learning addresses many of the challenges described in developing general-purpose robots. By training a single policy to reach many goals, the agent can reuse knowledge of reaching one goal to quickly generalize to reaching new goals or can plan over sub-goals to accomplish new tasks beyond those possible from simple generalization. Moreover, by training agents to set goals for themselves, agents can autonomously acquire goal-reaching capabilities without requiring a user to manually provide reward supervision. Lastly, by leveraging advances in representation learning, these methods can be applied directly to high-dimensional state spaces such as images. These benefits make goal-conditioned RL a promising field of study to develop general-purpose robots, but we first need to answer an important question: what exactly do we mean by "goal"?

There are a number of possible representations of a goal that an agent can use, and the

exact representation of a goal will impact how easily an agent can self-generate goals, how general its goal-directed capabilities are, and how flexibly an agent can reuse these capabilities to solve new tasks. For example, a robot that can manipulate objects can represent a goal as a fixed-length vector containing the desired pose of each object in the scene, but this representation has a number of drawbacks. This goal representation requires knowing which objects will be in the workspace and tracking their locations, which in practice will require external sensors or prior knowledge of the object identities. Another drawback is that this agent cannot set goals or plan sub-goals that involve manipulating new objects that are added to the scene, since the representation assumes a fixed set of objects in the scene. In general, any task-specific or scene-specific goal representation limits an agent's ability to learn general-purpose goal-reaching capabilities with minimal user involvement.

If we want to deploy robots in unstructured environments, with cluttered and unknown objects, then we need a goal representation that does not require additional sensors or prior knowledge. To obtain such a goal representation, this thesis studies how agents can autonomously set *goal states*, where a state is the raw sensory observation of the robot. For example, given a robot with a camera, the state is the current image of the scene, the goal is a specific goal image, and the robot's objective is to take actions in the world until its camera matches a desired goal image. By relying solely on the sensors of an agent to define the state and goal spaces, the methods in this thesis do not require prior knowledge about the environment and focus on enabling agents to set goals for themselves and plan directly in these general-purpose spaces.

**Challenges with goal-conditioned reinforcement learning.** There are a number of challenges with applying goal-conditioned reinforcement learning to real-world robot domains. For one, an agent in a new environment does not know the set of possible goals that it can set for itself. In other words, a vision-based robot can visualize many possible goal images, but not all of these goal images may be feasible. In order to enable agents to autonomously practice useful, realistic behaviors, we need a way for the agents to autonomously generate feasible goals for itself without using prior knowledge. Additionally, these agents must set useful goals for themselves that encourage them to explore the set of possible behaviors, rather than repeatedly practicing a small set of goals.

Another challenge is that an agent must also be able to reward itself for reaching goals without external supervision, but it is challenging to define a reward that both captures the semantics of a goal and that facilitates fast learning. For example, in vision based tasks, pixel-wise Euclidean distance to a goal image is not an effective reward since ancillary factors such as lighting conditions can easily dominate the reward signal, and rewards that check for exact equality are too sparse to learn from in these high dimensions.

Even if we do train a goal-conditioned policy, we also need to develop ways to reuse these goal-conditioned policies to accomplish new user-specified goals and tasks. Autonomously learning how to reach every possible goal from every possible start state can be extremely time-consuming and does not directly take advantage of the Markovian properties of goal

states. If an agent can move from state $A$ to state $B$ and from state $B$ to state $C$, then we would like to develop methods that enable this agent to immediately move from $A$ to $C$ without requiring additional practice. Furthermore, not all tasks can be specified by a single goal state, and it is unclear how a goal-reaching policy can be used to accomplish these other tasks.

**Outline.** This thesis presents methods for addressing the challenges outlined above with autonomously setting goals and reusing these goal-directed capabilities. We begin in Chapter 2 by providing background on goal-conditioned RL, describing the tools that we use, such as $Q$-learning and variational autoencoders. This chapter also presents our first algorithm for learning goal-conditioned policies, in which we derive the proper reward function for goal-conditioned reinforcement learning.

In Chapter 3, we then address the challenges associated with setting goals in complex state spaces. We discuss how agents can learn to generate feasible goals in new environments by learning generative models over the state space using previously observed states. We present a method that alternates between goal-directed exploration to observe new states and using these states to update the generative model. We also define what it means for an agent to perform optimal goal-directed exploration, propose a novel method for training the generative models, and present conditions under which our proposed method converges to the optimal goal-directed exploration.

Next, we address the challenges with reusing these goal-directed policies to solve new tasks. In Chapter 4, we discuss how goal-reaching requires implicitly learning about which goals can be reached, and how this knowledge can be reused to optimize arbitrary rewards or goals that are far from the current state by planning over feasible sub-goals. We also discuss how planning over high-dimensional state spaces such as images is difficult, both because it is computationally expensive and because it can lead to adversarial examples. We then present a method that plans in a learned, latent space, which implicitly constrains the optimization to the manifold of valid states.

In Chapter 5, we then study how the techniques in goal-conditioned reinforcement learning can be extended to a more general class of problems, such as learning to reach a distribution of states rather than a specific state, or meta-learning to maximize an arbitrary reward function. Lastly, we conclude in Chapter 6 with a discussion of the implications of this research.

# Chapter 2

# Goal-Conditioned Reinforcement Learning

In this chapter, we present the formalisms, notation, and objectives of goal-conditioned reinforcement learning. We will also present techniques that we will use throughout the thesis, as well as a method for training goal-conditioned policies assuming that we are given a specific goal. In future chapters, we will study how to generate these goals autonomously and reuse the resulting goal-conditioned policies to accomplish new tasks. We begin by presenting the standard reinforcement learning problem formulation.

## 2.1 Background

Reinforcement learning (RL) is concerned with reward maximization in a Markov decision process (MDP), defined by a tuple $(\mathcal{S}, \mathcal{A}, p_d, r, \rho_0, \gamma)$ [200], where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $p_d(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$ is the state dynamics transition model, $r(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ is the reward function, $\rho_0$ is the initial state distribution, and $\gamma \in [0, 1)$ is the discount factor. In each episode, the agent's initial state $\mathbf{s}_0 \in \mathcal{S}$ is sampled from an initial state distribution $\mathbf{s}_0 \sim \rho_0(\mathbf{s}_0)$, the agent chooses an action $\mathbf{a} \in \mathcal{A}$ according to a stochastic policy $\mathbf{a}_t \sim \pi(\cdot|\mathbf{s}_t)$, and the next state is sampled from the state transition dynamics $\mathbf{s}_{t+1} \sim p_d(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)$. We will use $\boldsymbol{\tau}$ to denote a trajectory sequence $(\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \dots)$ and denote sampling a trajectory as $\boldsymbol{\tau} \sim \pi$. The objective of an agent is to learn a policy that maximizes the sum of discounted rewards,

$$\mathbb{E}_{\boldsymbol{\tau} \sim \pi} \left[ \sum_{t=1}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \right].$$

**$Q$-learning** There are many algorithms for learning reward-maximizing policies, and a particularly important class of algorithms that can use off-policy data is the class of $Q$-learning algorithms. $Q$-learning algorithms learn $Q$-functions, which predicts the expected sum of

discounted reward, also called the expected *returns*, of a policy after taking action $\mathbf{a}_t$ in state $\mathbf{s}_t$. Formally, a $Q$-function is defined as

$$Q^\pi(\mathbf{s}, \mathbf{a}) \doteq \mathbb{E}_{\boldsymbol{\tau}\sim\pi}\left[\sum_{t=0}^{\infty} r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \mid \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}\right]$$

and can be written recursively [200] as

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\mathbf{s}'\sim p_d(\cdot|\mathbf{s},\mathbf{a})}\left[r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma \mathbb{E}_{\mathbf{a}'\sim\pi(\cdot|\mathbf{s}')}[Q^\pi(\mathbf{s}', \mathbf{a}')]\right]. \tag{2.1}$$

Similarly, the optimal $Q$-function can be defined recursively as following:

$$Q^*(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\mathbf{s}'\sim p_d(\cdot|\mathbf{s},\mathbf{a})}\left[r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma \max_{\mathbf{a}'} Q^*(\mathbf{s}', \mathbf{a}')\right]$$

The optimal policy can then recovered according to the indicator distribution $\pi(\mathbf{a} \mid \mathbf{s}) = \delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}))$, which deterministically selects the action that maximizes the $Q$-function at every state.

$Q$-learning algorithms train function approximators, such as neural networks, to approximate these $Q$-functions by estimating the right-hand side of the expectation in Equation (2.1) with a single sample. Specifically, let $Q_w$ be a function approximator with parameters $w$. Given a sample transition $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$ where the state $\mathbf{s}'$ is sampled from $p_d(\cdot \mid \mathbf{s}, \mathbf{a})$, $Q$-learning methods minimize the following squared Bellman error:

$$\mathcal{L}(w) = \|Q_w(\mathbf{s}, \mathbf{a}) - y_{\text{target}}\|^2, \tag{2.2}$$

where the target is tranditionally approximated with

$$y_{\text{target}} = r + \gamma \max_{\mathbf{a}'} Q_{\bar{w}}(\mathbf{s}', \mathbf{a}'). \tag{2.3}$$

For numerical instabilities, the target $Q$-value is typically approximated with a target network $Q_{\bar{w}}$ with parameter $\bar{w}$ that slowly tracks $w$.

Different algorithms different on how they handle the numerical instabilities [149, 138, 71] and how they approximate the maximization in Equation (2.3) [149, 138, 77, 84, 71], but all $Q$-learning algorithms take this form. $Q$-learning algorithms are important because they only require tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$ that can be collected by *any* policy, making it an off-policy algorithm. In other words, this data can be collected once and reused to train multiple $Q$-functions, an important property for learning policies that can reach multiple goals.

Lastly, we define a similar quantity, called the $V$-function, that represents the expected returns conditioned on the current state:

$$V^\pi(\mathbf{s}) = \mathbb{E}_{\boldsymbol{\tau}\sim\pi}\left[\sum_{t=0}^{\infty} r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \mid \mathbf{s}_0 = \mathbf{s}\right].$$

The $V$-function can also be written recursively as

$$V^\pi(\mathbf{s}) = \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s}),\mathbf{s}'\sim p_d(\cdot|\mathbf{s},\mathbf{a})}\left[r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma V^\pi(\mathbf{s}')\right]$$

and can be trained similarly to $Q$-functions using transitions $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$. To keep the notation uncluttered, we will often omit the dependence of $Q$ and $V$ on $\pi$.

### 2.1.1 Goal-conditioned reinforcement learning.

Goal-conditioned RL [104] is a type of RL problem in which the reward is parameterized by a specific goal that the agent wants to reach. Formally, the goal-conditioned RL problem augments the MDP with a goal space $\mathcal{G}$ and goal distribution $\rho_g$. Moreover, the reward function $r(\mathbf{s}, \mathbf{a}, \mathbf{s}', \mathbf{g})$ is conditioned on a goal $\mathbf{g}$.

At the beginning of each episode, a goal is sampled $\mathbf{g} \sim \rho_g$ along with the agent's initial state $\mathbf{s}_0 \sim \rho_0(\mathbf{s}_0)$. The agent then chooses an action according to a stochastic *goal-conditioned* policy $\mathbf{a}_t \sim \pi(\cdot|\mathbf{s}_t, \mathbf{g})$, and the next state is sampled from the state transition dynamics $\mathbf{s}_{t+1} \sim p_d(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)$. We note that the state transition dynamics are independent of the goal.

Typically, the goal space and reward function must be defined manually, with a common choice being to equate the goal and state space (i.e., $\mathcal{G} = \mathbf{s}$) and to use a sparse indicator reward $r(\mathbf{s}, \mathbf{g}) = \mathbb{I}\{\mathbf{s} = \mathbf{g}\}$. However, this approach presents a number of challenges both in theory and in practice. From a theoretical perspective, the indicator reward will almost surely equal zero for environments with continuous goal spaces and non-trivial stochastic dynamics. From a practical perspective, such sparse rewards can be slow to learn from, as most transitions provide no reward supervision, while manually designing dense reward functions that provide a better learning signal is time-consuming and often based on heuristics. In Section 2.3.1, we will present a framework that addresses these practical and theoretical considerations by casting goal-conditioned RL as probabilistic inference. Lastly, the choice of the goal distribution $\rho_g$ is often left to the user, but we will discuss in Chapter 3 how this goal distribution can be learned from data.

The advantage of the goal-conditioned formulation is that one does not need to train a separate policy for each task. By training a single policy to reach multiple goals, one enables a single policy to reuse knowledge about reaching some goals to quickly learn to reach new goals. As we will discuss in Chapter 4, these policies can then be reused to solve new tasks by planning over the appropriate sub-goals. Furthermore, one can reuse the same data across multiple goals by training goal-conditioned value functions with off-policy algorithms such as $Q$-learning, resulting in sample efficient learning.

**Goal-conditioned value functions.** Like a standard $Q$-function, a goal-conditioned $Q$-function $Q^\pi(\mathbf{s}, \mathbf{a}, \mathbf{g})$ learns the expected return for a given goal $\mathbf{g}$, when conditioned on taking action $\mathbf{a}$ at state $\mathbf{s}$. Similar to Equation (2.2), one can train an approximate goal-conditioned $Q$-function parameterized by $w$ by minimizing the squared Bellman error

$$\mathcal{L}(w) = \left\| Q_w(\mathbf{s}, \mathbf{a}, \mathbf{g}) - (r + \gamma \max_{\mathbf{a}'} Q_{\bar{w}}(\mathbf{s}', \mathbf{a}', \mathbf{g})) \right\|^2,$$

given a state $\mathbf{s}$, action $\mathbf{a}$, next state $\mathbf{s}'$, goal $\mathbf{g}$, and corresponding goal-conditioned reward $r$.

Because the transition dynamics $p_d$ does not depend on the goal and because $Q$-learning can use off-policy data, these value functions can be learned for any goal $\mathbf{g}$ using the same off-policy $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ tuple, provided that one can compute the reward $r(\mathbf{s}, \mathbf{a}, \mathbf{s}', \mathbf{g})$. Relabeling previously visited states with the reward for any goal leads to a natural data augmentation

strategy, since each tuple can be replicated many times for many different goals without additional data collection. Kaelbling [104] and Andrychowicz et al. [4] used this property to produce an effective curriculum for solving multi-goal task with delayed rewards. As we will discuss in Section 3.1, relabeling past experience with different goals enables goal-conditioned value functions to learn much more quickly from the same amount of data.

**Finite-horizon formulation.** In this work, we will also consider a finite-horizon, goal-conditioned Markov decision process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{G}, \mathcal{A}, p_d, r, T_{\max}, \rho_0, \rho_g)$, where the primary difference is that the discount factor $\gamma$ is replaced with a time horizon $T_{\max}$. The reward function and policy may both depend on the time. In this case, the objective is to obtain a possibly policy $\pi(\mathbf{a}_t \mid \mathbf{s}_t, \mathbf{g}, t)$ to maximize the expected sum of rewards $\mathbb{E}_{\boldsymbol{\tau}\sim\pi}[\sum_{k=0}^{t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}, \mathbf{s}_{t'+1}, \mathbf{g}, k)]$ over the remaining $t$ time steps.

The $Q$-function also varies with time and predicts the expected sum of future rewards, given the current state $\mathbf{s}$, goal $\mathbf{g}$, and remaining time $t$:

$$Q^{\pi}(\mathbf{s}, \mathbf{a}, \mathbf{g}, t) = \mathbb{E}\left[\sum_{t'=0}^{t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}, \mathbf{s}_{t'+1}, \mathbf{g}, t') \mid \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}, \pi \text{ is conditioned on } \mathbf{g}\right],$$

and similarly for the $V$-function. While both formulations have been used, this formulation is particularly useful for using goal-conditioned value functions as a form of temporal abstraction, as we will discuss in Chapter 4.

### 2.1.2 Variational Autoencoders.

A useful model of complex, multimodal data is a latent variable model. Latent variable models assume that data $\mathbf{x} \in \mathbb{R}^{d_x}$ is generated by first sampling a latent code $\mathbf{z} \in \mathbb{R}^{d_z}$ from a prior distribution $p(\mathbf{z})$ and then sampling $\mathbf{z}$ from a conditional distribution $p(\mathbf{x} \mid \mathbf{z})$. Together, the prior $p(\mathbf{z})$ and conditional distribution $p(\mathbf{x} \mid \mathbf{z})$ comprise the *generative model* of $\mathbf{x}$. Typically, the latent dimension, $d_z$, is much smaller than the observation dimension, $d_x$.

Latent variable models are useful because they incorporate prior knowledge that observation can be decomposed into simple, underlying factors. For example, if $\mathbf{x}$ is an image of a scene, $\mathbf{z}$ may represent the identity of different objects, their location, the pose of the camera, and other "latent" factors that are not directly recorded in the image $\mathbf{x}$ but that contribute to its generation. In this example, the distribution $p(\mathbf{x} \mid \mathbf{z})$ represents the (stochastic) way that those factors result in an image. One trains a generative model by fitting $p(\mathbf{z})$ and $p(\mathbf{x} \mid \mathbf{z})$ to data. By learning the generative model, one implicitly learns about how the world is organized and how to generate new, novel observations.

One can invert this generative process: given an observation $\mathbf{x}$, what were the latent variables $\mathbf{z}$ that produced $x$? Obtaining this posterior distribution, $p(\mathbf{z} \mid \mathbf{x})$, is called probabilistic inference. Probabilistic inference is useful because it decomposes a scene into its underlying factors of variation. In Chapter 3, we will use these abilities to represent and generate goals for an autonomous agent.

Given a dataset of observations, $\mathcal{D}$, a generative model is typically trained with maximum likelihood estimation

$$\max_{\theta} \sum_{x \in \mathcal{D}} \log p_\theta(x). \tag{2.4}$$

However, with latent variable models, computing the marginal distribution $p(x)$ requires marginalizing over $\mathbf{z}$, as in $p(\mathbf{x}) = \int_{\mathcal{Z}} p(\mathbf{z})p(\mathbf{x} \mid \mathbf{z})d\mathbf{z}$. Computing this integral and optimizing it is intractable for many latent variable models, and we avoid this intractability by training variational autoencoders with variational inference.

Variational autoencoders [116, 182] are a special type of latent variable model that can be tractably trained even when using expressive neural networks to represent the generative model. For the generative model, VAE methods use a fixed prior $p(\mathbf{z})$ that usually corresponds to a standard Gaussian distribution in $\mathbb{R}^{d_z}$ and train a neural network with parameter $\theta$ to represent the conditional distribution $p_\theta(\mathbf{x} \mid \mathbf{z})$. For probabilistic inference, VAE methods perform amortized inference by training a separate neural network with parameter $\phi$ to represent the posterior of $\mathbf{z}$ given an observation $\mathbf{x}$, $q_\phi(\mathbf{z} \mid \mathbf{x})$.

Rather than optimizing Equation (2.4), VAEs maximize a lower bound on the log-likelihood of the data. This lower bound is called the evidence lower bound, or ELBO, and for a given observation $\mathbf{x}$ is

$$\text{ELBO}(\mathbf{x}) = \mathbb{E}_{enc(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} \mid \mathbf{z})] - D_{\text{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \middle|\middle| p(\mathbf{z})\right) \leq \log p_\theta(x)$$

where $D_{\text{KL}}$ is the KL divergence.

In practice, the encoder and decoder parameters, $\phi$ and $\theta$ respectively, are jointly trained to maximize

$$\mathcal{L}(\theta, \phi; \mathbf{x}) = \mathbb{E}_{enc(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} \mid \mathbf{z})] - \beta D_{\text{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \middle|\middle| p(\mathbf{z})\right), \tag{2.5}$$

where $\beta$ is a hyperparameter that has been shown to encourage the generative model to learn representations that are more disentangled when $\beta > 1$ [94]. The encoder $q_\phi$ typically parameterizes the mean and log-variance diagonal of a Gaussian distribution, i.e., $q_\phi(\mathbf{x}) = \mathcal{N}(\mu_\phi(\mathbf{x}), \sigma^2_\phi(\mathbf{x}))$, where $\mu_\phi$ and $\sigma^2_\phi$ are neural networks. The decoder $p_\theta$ typically parameterizes a Gaussian or the parameters of a Bernoulli distribution for each dimension of $\mathbf{x}$.

## 2.2 Related Work

Traditionally, the focus of goal-conditioned reinforcement learning literature has been on training goal-conditioned policies, assuming that a goal and goal-conditioned reward is provided during exploration [104, 190, 4, 177, 171]. The methods that we present build off of these prior works to train goal-conditioned policies. For example, in Section 3.1 we present a

method for retroactively relabelling data with goals sampled from a learned goal distribution. Prior work found that goal relabeling accelerates learning [104, 4, 177, 136], and we find that our specific relabeling strategy further improves the rate of learning.

However, these prior works do not study the question of where these goals and rewards come from, which in practice must be provided externally be the environment designer. As a results, the core ideas of this thesis are largely complementary to these methods: rather than only proposing a better method for training goal-reaching policies, we focus on developing methods that enable agents to autonomously set goals and provide rewards for themselves, as well as developing methods for reusing the learned goal-reaching policies. Below, we discuss the relationship to prior work in the context of autonomous learning and skill reuse, and also conclude with a discussion to related fields and alternative problem formulations.

**Exploration and goal setting.** The methods that we describe in Chapter 3 learn without any task rewards, and instead focus on developing agents that can autonomously learn about the environment by setting and reaching goals. This autonomous learning is related to exploration methods, which modify the reward based on state visitation frequency to encourage a policy to visit novel states [13, 163, 203, 29, 141, 199, 166, 23, 22, 150, 203, 68]. While these exploration methods can also be used without a task reward, they provide no mechanism for distilling the knowledge gained from visiting diverse states into flexible policies that can be applied to accomplish new goals at test-time: their policies visit novel states, and they quickly forget about them as other states become more novel. Similarly, methods that provably maximize state entropy without using goal-directed exploration [91] or methods that define new rewards to capture measures of intrinsic motivation [150] and reachability [188] do not produce reusable policies.

Other prior methods do train goal-reaching policies, but propose to choose goals based on heuristics such as learning progress [8, 214, 35], how off-policy the goal is [152], level of difficulty [93], or likelihood ranking [231]. In contrast, in Section 3.2 we provide a principled framework that optimizes a concrete and well-motivated exploration objective, that can provably maximize this objective under regularity assumptions, and that empirically outperforms many of these prior work.

**Model-based reinforcement learning** In Chapter 4, we discuss how to use goal-conditioned policies as the abstraction for planning in order to handle tasks with a longer horizon. This planning is similar to the planning that occurs in model-based reinforcement learning. Model-based reinforcement learning methods use a learned dynamics model to plan over a sequence of actions [175, 132, 154, 33]. However, when the observations are high-dimensional, such as images, model errors for direct prediction compound quickly, making model-based planning difficult [62, 53, 24, 54, 105]. Rather than planning directly over states, we study how to plan at a temporally-abstract level by using state embeddings and goal-conditioned policies.

A number of past works have also studied embedding high-dimensional observations into a low-dimensional latent space for planning [218, 64, 86, 122, 229]. While our method also plans

in a latent space, we additionally use a model-free goal-conditioned policy as the abstraction to plan over, allowing our method to plan over temporal abstractions rather than only state abstractions.

Planning over subgoals for a low-level goal-reaching policy also bears a resemblance to hierarchical RL, where prior methods have used model-free learning on top of goal-conditioned policies [41, 220, 47, 215, 136, 74, 152]. By using a planner rather than a learned policy at the higher level, our method can flexibly plan to solve new tasks and benefit from the compositional structure of planning.

Planning with goal-conditioned value functions has also been studied when there are a discrete number of goals [124] or skills [2], in which case graph-search algorithms can be used to plan [59, 139, 56]. In Chapter 4, we not only provide a concrete instantiation of planning with goal-conditioned value functions, but we also present a new method for scaling this planning approach to images, which reside in a lower-dimensional manifold.

**Representation learning.** Like our work, a number of prior works have used unsupervised learning to acquire better representations for RL. These methods use the learned representation as a substitute for the state for the policy, but require additional information, such as access to the ground truth reward function based on the true state during training time [95, 82, 218, 64, 126, 103], expert trajectories [198], human demonstrations [195], or pre-trained object-detection features [128]. In contrast, we learn to generate goals and use the learned representation to obtain a reward function for those goals without any of these extra sources of supervision. Many prior works have also focused on learning controllable and disentangled representations [192, 27, 30, 181, 44, 204]. We use a method based on variational autoencoders, but these prior techniques are complementary to ours and could be incorporated into our method.

**Contextual reinforcement learning.** Goal-conditioned reinforcement learning is a special case of contextual reinforcement learning, in which a policy is conditioned on some variable $\mathbf{z}$ in addition to the state $\mathbf{s}$. Contextual policies has been studied in the form of latent-variable-conditioned policies, where latent variables are interpreted as options [201] or abstract skills [90, 81, 58, 79, 66]. The resulting skills are diverse, but have no grounded interpretation, while goal-reaching policies can be used immediately after unsupervised training to reach diverse user-specified goals. Contextual policies have also been studied in the work on successor features [120, 10, 18, 11, 76], which assumes that reward have the form $r(\mathbf{s}, \mathbf{a}) = \phi(\mathbf{s})^T w$ and conditions policies on different values of $w$. These methods rely on manually specified state features $\phi$, whereas our work directly learning to reach goal states, removing the need for any manual feature engineering.

In Section 5.1, we discuss a fully general class of contextual reinforcement learning problem, in which we can represent arbitrary reward functions by specify a task with a goal distribution learned from a *set* of example goal states. A number of prior methods learn rewards [1, 208, 70] or policies [207, 195, 140, 55] using expert trajectories or observations. Many of these prior

methods require state sequences from expert demonstrations [1, 207, 195, 140, 55], or focus on solving single tasks or goal-reaching tasks [70]. In contrast, we will present a contextual framework in which *arbitrary* rewards can be represented as goal distributions, and only requires observations of successful outcomes to fit the goal distribution.

In Section 5.2, we discuss how this connection to contextual policies also relates our work to offline meta-reinforcement learning algorithms. Specifically, we will present semi-supervised meta-actor-critic (SMAC), a method that uses a context-based adaptation procedure similar to Rakelly et al. [176].

Many prior meta-RL algorithms assume that reward labels are provided with each episode of online interaction [52, 65, 80, 224, 89, 176, 99, 118, 235, 225, 233, 108]. In contrast to these prior methods, our method only requires offline prior data with rewards, and additional online interaction does not require any ground truth reward signal.

Prior works have also studied other formulations that combine unlabeled and labeled trials. For example, imitation and inverse reinforcement learning methods use offline demonstrations to either learn a reward function [1, 63, 96, 69] or to directly learn a policy [189, 184, 96, 180, 167]. Semi-supervised and positive-unlabeled reward learning [222, 236, 119] methods use reward labels provided for some interactions to train a reward function for RL. However, all of these methods have been studied in the context of a single task. While these methods focus on recovering a policy that maximizes a specific reward function, we focus on meta-learning an RL procedure that can adapt to new reward functions. In other words, we do not focus on recovering a single reward function, because there is no single test time reward or task. Instead, we focus on generating reward labels for meta-training that mitigate the distribution shift between the offline data and online data at test-time.

Our method addresses a similar problem to prior offline meta-RL methods [148, 50], but we show that these approaches generally underperform in low-data regimes, whereas our method addresses the distribution shift problem by using online interactions without requiring additional reward supervision. In our experiments, we found that SMAC greatly improves the performance on both training and held-out tasks. Lastly, SMAC is also related to unsupervised meta-learning methods [79, 101], which annotate data with their own rewards. In contrast to these methods, we assume that there exists an offline dataset with reward labels that we can use to learn to generate similar rewards.

**Alternative formulations to control.** Lastly, we note that reward maximization is not the only formulation for learning goal-reaching behaviors. Stochastic control [28, 75, 98, 223] is a closely related problem of finding control laws that allow agents to move from an initial state to some desired goal state while incurring minimal cost. Rawlik et al. [179] consider a continuous-time setting and propose an expectation maximization algorithm that assumes linear Gaussian dynamics, while more recent work has explored finding control laws for non-linear stochastic system dynamics [183, 226]. In contrast to this strand of research, this thesis studies a discrete-time setting and does not make assumptions or assumes knowledge of the system dynamics but only requires the ability to interact with the environment to

learn an outcome-driven policy.

Another formulation for learning goal-reaching policies is that of probabilistic inference. Under this framework, learning a policy corresponds to inferring the distribution of actions, conditioned on reaching a goal in the future. Several prior works cast RL and control as probabilistic inference [61, 70, 97, 133, 178, 196, 205, 234] or KL divergence minimization [111, 169], with the aim of reformulating standard reward-based RL, assuming that the reward function is given. In contrast, our work removes the need to manually specify a task-specific reward or likelihood function, by learning a reward function from data, based on derivations discussed in Section 2.3.

Past work has also studied this relationship between control and probabilistic inference in the context of reaching a goal or desired outcome [6, 70, 97, 210]. However, these past work have limitations that make it challenging to apply the methods to more complex domains in a sample-efficient manner. Toussaint et al. [211] and Hoffman et al. [97] focus on exact inference methods that require time-varying tabular or time-varying Gaussian value functions, and the work in Fu et al. [70] requires on-policy trajectory samples, which can be expensive or time-consuming to collect.

In the next chapter, we present a variational inference method that eliminates the need to train a time-varying value function, and enables us to use expressive neural networks to represent an approximate value function, making our method applicable to high-dimensional, continuous, and non-linear domains. Moreover, in contrast to the work in Fu et al. [70], we will derive an off-policy method by introducing a variational distribution $q_T$ over the time when the outcome is reached.

## 2.3 Goal-Reaching as Probabilistic Inference

Standard goal-conditioned RL provides an appealing formalism for automated learning of goal-reaching skills, but in practice, it requires considerable care and manual design. One particularly delicate decision is the design of the reward function, which has a significant impact on the resulting policy but is largely heuristic in practice, often lacks theoretical grounding, can make effective learning difficult, and may lead to reward mis-specification.

To avoid these shortcomings, we propose to circumvent the process of manually specifying a reward function altogether: Instead of framing the goal-conditioned reinforcement learning problem as finding a policy that maximizes a heuristically-defined reward function, we express it probabilistically, as inferring a state–action trajectory distribution conditioned on a desired future outcome. By building off of prior work on probabilistic perspectives on RL [61, 109, 178, 209, 210, 234] and goal-directed RL in particular [6, 32, 97, 211], we derive a tractable variational objective, an temporal-difference algorithm that provides a shaping-like effect for effective learning, as well as a reward function that captures the semantics of the underlying decision problem and facilitates effective learning.

We demonstrate that unlike prior works that proposed inference methods for finding policies that achieve desired outcomes [6, 70, 97, 211], the resulting algorithm, outcome-

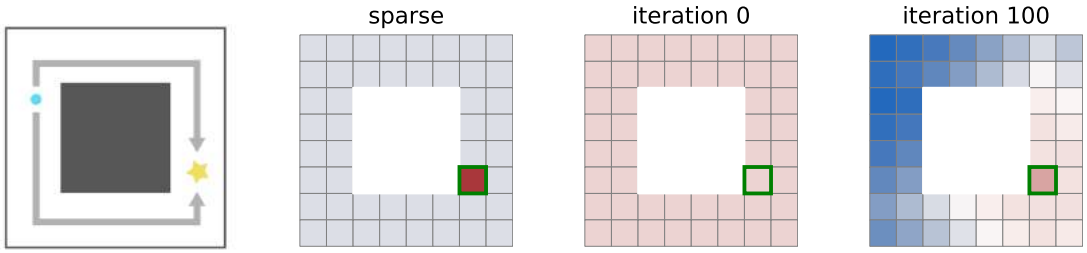



**Figure 2.1:** Illustration of the shaping effect of the reward function derived from the goal-directed variational inference objective. (Left) A 2-dimensional grid world with a desired outcome marked by a star. (Middle left) The corresponding sparse reward function provides little shaping. (Middle right) The reward function derived from our variational inference formulation at initialization. (Right) The derived reward function after training. We see that the derived reward learns to provide a dense reward signal everywhere in the state space.

driven actor–critic (ODAC), is amenable to off-policy learning and applicable to complex, high-dimensional continuous control tasks over finite and infinite horizons. The resulting variational algorithm can be interpreted as an automatic shaping method, where each iteration learns a reward function that automatically provides dense rewards, as we visualize in Figure 2.1. In tabular settings, ODAC is guaranteed to converge to an optimal policy, and in non-tabular settings with linear Gaussian transition dynamics, the derived optimization objective is convex in the policy, facilitating easier learning. In high-dimensional and non-linear domains, our method can be combined with deep neural network function approximators to yield a deep reinforcement learning method that does not require manual specification of rewards, and leads to good performance on a range of benchmark tasks.

The core contribution of this section is the probabilistic formulation of a general framework for inferring policies that lead to desired outcomes. We show that this formulation gives rise to a variational objective from which we derive a novel outcome-driven Bellman backup operator with a shaping-like effect that ensures a clear and dense learning signal even in early stages of training. Crucially, this "shaping" emerges automatically from a variational lower bound, rather than the heuristic approach for incorporating shaping that is often used in standard RL. We demonstrate that the resulting variational objective leads to an off-policy temporal-difference algorithm and evaluate it on a range of reinforcement learning tasks without having to manually specify task-specific reward functions. In our experiments, we find that our method results in significantly faster learning across a variety of robot manipulation and locomotion tasks than alternative approaches.

### 2.3.1 Outcome-Driven Reinforcement Learning

In this section, we derive a variational inference objective to infer an approximate posterior policy for achieving desired outcomes. Instead of using the heuristic goal-reaching rewards discussed in Section 2.1, we will derive a general framework for *inferring actions that lead*

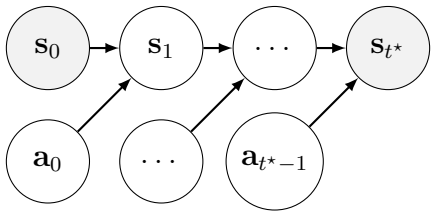



**Figure 2.2:** A probabilistic graphical model of a state–action trajectory with observed random variables $\mathbf{s}_0$ and $\mathbf{s}_{t^\star}$.

*to desired outcomes* by formulating a probabilistic objective, using the tools of variational inference. As we will show in the following sections, we use this formulation to translate the problem of inferring a policy that leads to a desired outcome into a tractable variational optimization problem, which we show corresponds to an RL problem with a well-shaped, dense reward signal from which the agent can learn more easily.

We start with a warm-up problem that demonstrates how to frame the task of achieving a desired outcome as an inference problem in a simplified setting. We then describe how to extend this approach to more general settings. Finally, we show that the resulting variational objective can be expressed as a recurrence relation, which allows us to derive an outcome-driven variational Bellman operator and prove an outcome-driven probabilistic policy iteration theorem.

### 2.3.2 Warm-up: Achieving a Desired Outcome at a Fixed Time Step

We first consider a simplified problem, where the desired outcome is to reach a specific state $\mathbf{g} \in \mathcal{S}$ at a specific time step $t^\star$ when starting from initial state $\mathbf{s}_0$. We can think of the starting state $\mathbf{s}_0$ and the desired outcome $\mathbf{g}$ as boundary conditions, and the goal is to learn a stochastic policy that induces a trajectory from $\mathbf{s}_0$ to $\mathbf{g}$. To derive a control law that solves this stochastic boundary value problem, we frame the problem probabilistically, as inferring a state–action trajectory posterior distribution *conditioned on the desired outcome and the initial state*. We will show that, by framing the learning problem this way, we obtain an algorithm for learning outcome-driven policies without needing to manually specify a reward function. We consider a model of the state–action trajectory up to and including the desired outcome $\mathbf{g}$,

$$p_{\boldsymbol{\tau}_{0:t},\mathbf{s}_{t+1}}(\boldsymbol{\tau}_{0:t}, \mathbf{g} \mid \mathbf{s}_0) \doteq p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) p(\mathbf{a}_t \mid \mathbf{s}_t) \prod_{t'=0}^{t-1} p_d(\mathbf{s}_{t'+1} \mid \mathbf{s}_{t'}, \mathbf{a}_{t'}) p(\mathbf{a}_{t'} \mid \mathbf{s}_{t'}),$$

where $t \doteq t^\star - 1$, $p(\mathbf{a}_t \mid \mathbf{s}_t)$ is a conditional action prior, and $p_d(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ is the environment's state transition distribution. If the dynamics are simple (e.g., tabular or Gaussian), the

posterior over actions can be computed in closed form [6], but we would like to be able to infer outcome-driven posterior policies in any environments, including those where exact inference may be intractable. To do so, we start by expressing posterior inference as the variational minimization problem

$$\min_{q \in \mathcal{Q}} D_{\mathrm{KL}}\left(q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid \mathbf{s}_0) \middle\| p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid \mathbf{s}_0, \mathbf{s}_{t^\star} = \mathbf{g})\right), \tag{2.6}$$

where $\tilde{\boldsymbol{\tau}}_{0:t}$ is the state–action trajectory up to $t^\star$, but excluding $\mathbf{s}_0$ and $\mathbf{s}_{t^\star}$, $D_{\mathrm{KL}}\left(\cdot \middle\| \cdot\right) D_{\mathrm{KL}}(\cdot \,\|\, \cdot)$ is the KL divergence, and $\mathcal{Q}$ denotes the variational family over which to optimize. We consider a family of distributions parameterized by a policy $\pi$ and defined by

$$q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0) \doteq \pi(\mathbf{a}_t \mid \mathbf{s}_t) \prod_{t'=0}^{t-1} p_d(\mathbf{s}_{t'+1} \mid \mathbf{s}_{t'}, \mathbf{a}_{t'}) \pi(\mathbf{a}_{t'} \mid \mathbf{s}_{t'}), \tag{2.7}$$

where $\pi \in \Pi$, a family of policy distributions, and where $\prod_{t=0}^{t-1} p_d(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ is the true action-conditional state transition distribution up to but excluding the state transition at $t^\star - 1$, since $\mathbf{s}_{t^\star} = \mathbf{g}$ is observed. Under this variational family, the inference problem in Equation (2.6) can be equivalently stated as the problem of maximizing the following objective with respect to the policy $\pi$:

**Proposition 1.** *Given $q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0)$ from Equation (2.7), any state $\mathbf{s}_0$, termination time $t^\star$, and outcome $\mathbf{g}$, solving Equation (2.6) is equivalent to maximizing this objective with respect to $\pi \in \Pi$:*

$$\bar{\mathcal{F}}(\pi, \mathbf{s}_0, \mathbf{g}) \doteq \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}_{0:t} \mid \mathbf{s}_0)}\left[\log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) - \sum_{t'=0}^{t-1} D_{\mathrm{KL}}(\pi(\cdot \mid \mathbf{s}_{t'}) \,||\, p(\cdot \mid \mathbf{s}_{t'}))\right]. \tag{2.8}$$

*Proof.* See Appendix B.1.1. □

A variational problem of this form—which corresponds to finding a posterior distribution over actions)—can equivalently be viewed as a reinforcement learning problem:

**Corollary 1.** *The objective in Equation (2.8) corresponds to KL-regularized reinforcement learning with a time-varying reward function given by $r(\mathbf{s}_{t'}, \mathbf{a}_{t'}, \mathbf{g}, t') \doteq \mathbb{I}\{t' = t\} \log p_d(\mathbf{g} \mid \mathbf{s}_{t'}, \mathbf{a}_{t'})$*

Corollary 1 illustrates how a reward function *emerges automatically* from a probabilistic framing of outcome-driven reinforcement learning problems where the sole specification is which variable ($\mathbf{s}_{t^\star}$) should attain which value ($\mathbf{g}$). In particular, Corollary 1 suggests that we ought to learn the environment's state-transition distribution, and view the log-likelihood of achieving the desired outcome given a state–action pair as a "reward" that can be used for off-policy learning as described in Section 2.1. Importantly—and unlike in model-based

RL—such a transition model would not have to be accurate beyond single-step predictions, as it would not be used for planning (see Appendix B.3). Instead, $\log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t)$ only needs to be well shaped, which we expect to happen for commonly used model classes. For example, when the dynamics are linear-Gaussian, using a conditional Gaussian model [154] yields a reward function that is quadratic in $\mathbf{s}_{t+1}$, making the objective convex and thus more amenable to optimization.

### 2.3.3 Outcome-Driven Reinforcement Learning as Variational Inference

Thus far, we assumed that the time at which the outcome is achieved is given. In many problem settings, we do not know (or care) when an outcome is achieved. In this section, we present a variational inference perspective on achieving desired outcomes in settings where *no reward function* and *no termination time* are given, but only a desired outcome is provided. As in the previous section, we derive a variational objective and show that a principled algorithm and reward function emerge automatically when framing the problem as variational inference.

To derive such an objective, we modify the probabilistic model used in the previous section to model that the time at which the outcome is achieved is not known. As before, we define an "outcome" as a point in the state space, but instead of defining the event of "achieving a desired outcome" as a realization $\mathbf{s}_{t^\star} = \mathbf{g}$ for a known $t^\star$, we define it as a realization $\mathbf{s}_{T^\star} = \mathbf{g}$ for an *unknown* termination time $T^\star$ at which the outcome is achieved. Specifically, we model the distribution over the trajectory and the unknown termination time with

$$p_{\tilde{\boldsymbol{\tau}}_{0:T}, \mathbf{s}_T, T}(\tilde{\boldsymbol{\tau}}_{0:t}, \mathbf{g}, t \mid \mathbf{s}_0) = p_T(t) p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) p(\mathbf{a}_t \mid \mathbf{s}_t) \prod_{t'=0}^{t-1} p_d(\mathbf{s}_{t'+1} \mid \mathbf{s}_{t'}, \mathbf{a}_{t'}) p(\mathbf{a}_{t'} \mid \mathbf{s}_{t'}), \tag{2.9}$$

where $p_T(t)$ is the probability of reaching the outcome at $t+1$. Since the trajectory length is itself a random variable, the joint distribution in Equation (2.9) is a *transdimensional* distribution defined on $\biguplus_{t=0}^{\infty} \{t\} \times \mathcal{S}^{t+1} \times \mathcal{A}^{t+1}$ [97].

Unlike in the warm-up, the problem of finding an outcome-driven policy that eventually achieves the desired outcome corresponds to finding the posterior distribution over state–action trajectories *and* the termination time $T$ conditioned on the desired outcome $\mathbf{s}_T$ and a starting state. Analogously to Section 2.3.2, we can express this inference problem variationally as

$$\min_{q \in \mathcal{Q}} D_{\mathrm{KL}} \left( q_{\tilde{\boldsymbol{\tau}}_{0:t}, T}(\cdot \mid \mathbf{s}_0) \middle\| p_{\tilde{\boldsymbol{\tau}}_{0:t}, T}(\cdot \mid \mathbf{s}_0, \mathbf{s}_{T^\star} = \mathbf{g}) \right), \tag{2.10}$$

where $t$ denotes the time immediately *before* the outcome is achieved and $\mathcal{Q}$ denotes the variational family. In general, solving this variational problem in closed form is challenging, but by choosing a variational family $q_{\tilde{\boldsymbol{\tau}}_{0:T}, T}(\tilde{\boldsymbol{\tau}}_{0:t}, t \mid \mathbf{s}_0) = q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0) q_T(t)$, where $q_T$ is a

distribution over $T$ in some variational family $\mathcal{Q}_T$ parameterized by

$$q_T(t \mid \mathbf{s}_0) = q_{\Delta_{t+1}}(\Delta_{t+1} = 1) \prod_{t'=1}^{t} q_{\Delta_{t'}}(\Delta_{t'} = 0), \tag{2.11}$$

with Bernoulli random variables $\Delta_t$ denoting the event of "reaching $\mathbf{g}$ at time $t$ given that $\mathbf{g}$ has not yet been reached by time $t-1$," we can equivalently express the variational problem in Equation (2.10) in a way that is tractable and amenable to off-policy optimization:

**Theorem 1.** *Let $q_T(t)$ and $q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0)$ be as defined before, and define*

$$V^\pi(\mathbf{s}_t, \mathbf{g}; q_T) \doteq \mathbb{E}_{\pi(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T)] - D_{\mathrm{KL}}(\pi(\cdot \mid \mathbf{s}_t) \,\|\, p(\cdot \mid \mathbf{s}_t)) \tag{2.12}$$

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T) \doteq r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) + q_{\Delta_{t+1}}(\Delta_{t+1} = 0)\mathbb{E}_{p_d(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}[V^\pi(\mathbf{s}_{t+1}, \mathbf{g}; \pi, q_T)] \tag{2.13}$$

$$r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) \doteq q_{\Delta_{t+1}}(\Delta_{t+1} = 1) \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) - D_{\mathrm{KL}}\left(q_{\Delta_{t+1}} \middle\| p_{\Delta_{t+1}}\right). \tag{2.14}$$

*Then given any initial state $\mathbf{s}_0$ and outcome $\mathbf{g}$,*

$$D_{\mathrm{KL}}(q_{\tilde{\boldsymbol{\tau}}_{0:t},T}(\cdot \mid \mathbf{s}_0) \,\|\, p_{\tilde{\boldsymbol{\tau}}_{0:t},T}(\cdot \mid \mathbf{s}_0, \mathbf{s}_{T^\star} = \mathbf{g})) = -V^\pi(\mathbf{s}_0, \mathbf{g}; q_T) + \log p(\mathbf{g} \mid \mathbf{s}_0),$$

*where $\log p(\mathbf{g} \mid \mathbf{s}_0)$ is independent of $\pi$ and $q_T$ and hence maximizing $V^\pi(\mathbf{s}_0, \mathbf{g}; \pi, q_T)$ is equivalent to minimizing Equation (2.10).*

*Proof.* See Appendix B.1.2. □

This theorem tells us that the maximizer of $V^\pi(\mathbf{s}_t, \mathbf{g}; q_T)$ is equal to the minimizer of Equation (2.10). In other words, Theorem 1 presents a variational objective with dense reward functions defined solely in terms of the desired outcome and the environment dynamics, which we can learn directly from environment interactions. Thanks to the recursive expression of the variational objective, we can find the optimal variational over $T$ as a function of the current policy and $Q$-function analytically:

**Proposition 2.** *The optimal distribution $q_T^\star$ with respect to Equation (2.12) is*

$$q^\star_{\Delta_{t+1}}(\Delta_{t+1} = 0; \pi, Q^\pi) = \sigma\left(\Lambda(\mathbf{s}_t, \pi, q_T, Q^\pi) + \sigma^{-1}\left(p_{\Delta_{t+1}}(\Delta_{t+1} = 0)\right)\right), \tag{2.15}$$

*where*

$$\Lambda(\mathbf{s}_t, \pi, q_T, Q^\pi) \doteq \mathbb{E}_{\pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})p_d(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}, \mathbf{g}; q_T) - \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t)]$$

*and $\sigma(\cdot)$ is the sigmoid function, that is, $\sigma(x) = \frac{1}{e^{-x}+1}$ and $\sigma^{-1}(x) = \log \frac{x}{1-x}$.*

*Proof.* See Appendix B.1.3 □

In the next section, we discuss how we can learn the $Q$-function in Theorem 1 using off-policy transitions by deriving a temporal-difference algorithm for this problem.

### 2.3.4 Outcome-Driven Reinforcement Learning

In this section, we show that the variational objective in Theorem 1 is amenable to off-policy learning and that it can be estimated efficiently from single-step transitions. We then describe how to instantiate the resulting outcome-driven algorithm in large environments where function approximation is necessary.

#### 2.3.4.1 Outcome-Driven Policy Iteration

To develop an outcome-directed off-policy algorithm, we define the following Bellman operator:

**Definition 1.** *Given a function $Q : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, define the operator $\mathcal{T}^\pi$ as*

$$\mathcal{T}^\pi Q(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T) \doteq r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) + q_{\Delta_{t+1}}(\Delta_{t+1} = 0)\mathbb{E}\big[V(\mathbf{s}_{t+1}, \mathbf{g}; q_T)\big], \tag{2.16}$$

*where $r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta)$ is from Theorem 1, the expectation is w.r.t. $p_d(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$, and*

$$V(\mathbf{s}_t, \mathbf{g}; q_T) \doteq \mathbb{E}_{\pi(\mathbf{a}_t|\mathbf{s}_t)}\left[Q(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T)\right] + D_{\mathrm{KL}}(\pi(\cdot \mid \mathbf{s}_t) \,\|\, p(\cdot \mid \mathbf{s}_t)) \tag{2.17}$$

Unlike the standard Bellman operator, the above operator has a varying weight factor $q_{\Delta_{t+1}}(\Delta_{t+1} = 0)$, with the optimal weight factor given by Equation (2.15). From Equation (2.15), we see that as the outcome likelihood $p_d(\mathbf{g} \mid \mathbf{s}, \mathbf{a})$ becomes large relative to the $Q$-function, the weight factor automatically adjusts the target to rely more on the rewards.

Below, we show that repeatedly applying the operator $\mathcal{T}^\pi$ (policy evaluation) and optimizing $\pi$ with respect to $Q^\pi$ (policy improvement) converges to a policy that maximizes the objective in Theorem 1.

**Theorem 2.** *Assume MDP is ergodic and $|\mathcal{A}| < \infty$.*

1. *Outcome-Driven Policy Evaluation (ODPE): Given policy $\pi$ and a function $Q^0 : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, define $Q^{i+1} = \mathcal{T}^\pi Q^i$. Then the sequence $Q^i$ converges to the lower bound in Theorem 1.*

2. *Outcome-Driven Policy Improvement (ODPI): The policy that solves*

$$\pi^+ = \arg\max_{\pi' \in \Pi} \{\mathbb{E}_{\pi'(\mathbf{a}_t|\mathbf{s}_t)}\left[Q^\pi(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T)\right] - D_{\mathrm{KL}}(\pi'(\cdot \mid \mathbf{s}_t) \,||\, p(\cdot \mid \mathbf{s}_t)\} \tag{2.18}$$

   *and the variational distribution over $T$ defined in Equation (2.15) improve the variational objective, i.e., $\mathcal{F}(\pi^+, q_T, \mathbf{s}_0) \geq \mathcal{F}(\pi, q_T, \mathbf{s}_0)$ and $\mathcal{F}(\pi, q_T^+, \mathbf{s}_0) \geq \mathcal{F}(\pi, q_T, \mathbf{s}_0)$ for all $\mathbf{s}_0, \pi, q_T$.*

3. *Alternating between ODPE and ODPI converges to a policy $\pi^\star$ and a variational distribution over $T$, $q_T$, such that $Q^{\pi^\star}(\mathbf{s}, \mathbf{a}, \mathbf{g}; q_T^\star) \geq Q^\pi(\mathbf{s}, \mathbf{a}, \mathbf{g}; q_T)$ for all $(\pi, q_T) \in \Pi \times \mathcal{Q}_T$ and any $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$.*

*Proof.* See Appendix B.2. □

**Algorithm 1** ODAC: Outcome-Driven Actor–Critic

1: Initialize policy $\pi_\theta$, replay buffer $\mathcal{R}$, $Q$-function $Q_\phi$, and dynamics model $p_\psi$.
2: **for** iteration $i = 1, 2, \ldots$ **do**
3: Collect on-policy samples to add to $\mathcal{R}$ by sampling $\mathbf{g}$ from environment and executing $\pi$.
4: Sample batch $(\mathbf{s}, \mathbf{a}, \mathbf{s}', \mathbf{g})$ from $\mathcal{R}$.
5: Compute approximate reward and optimal weights with Equation (2.22) and Equation (2.15).
6: Update $Q_\phi$ with Equation (2.19), $\pi_\theta$ with Equation (2.20), and $p_\psi$ with Equation (2.21).
7: **end for**

This result tells us that alternating between applying the outcome-driven Bellman operator in Definition 1 and optimizing the bound in Theorem 1 using the resulting $Q$-function, which can equivalently be viewed as expectation maximization, will lead to a policy that induces an outcome-driven trajectory and solves the inference problem in Equation (2.10). As we discuss in Appendix B.2, this implies that Variational Outcome-Driven Policy Iteration is theoretically at least as good as or better than standard policy iteration for KL-regularized objectives.

#### 2.3.4.2 Outcome-Driven Actor–Critic (ODAC)

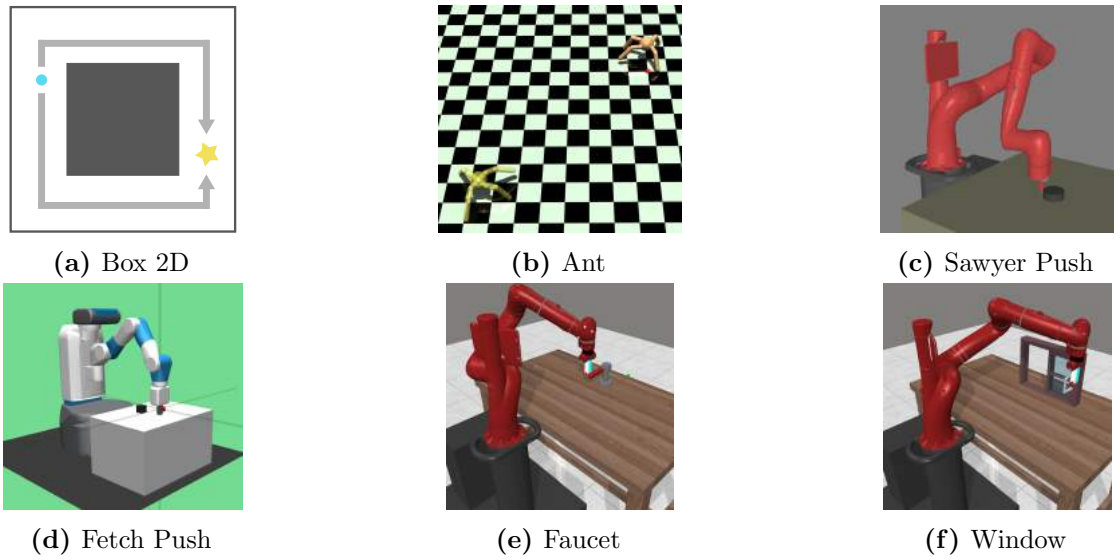
We now build on previous sections to develop a practical algorithm that handles large and continuous domains. In such domains, the expectation in the Bellman operator in Definition 1 is intractable, and so we approximate the policy $\pi_\theta$ and $Q$-function $Q_\phi$ with neural networks parameterized by parameters $\theta$ and $\phi$, respectively. We train the $Q$-function to minimize

$$\mathcal{F}_Q(\phi) = \mathbb{E}\left[\left(Q_\phi(\mathbf{s}, \mathbf{a}, \mathbf{g}) - (r(\mathbf{s}, \mathbf{a}, \mathbf{g}; q_\Delta) + q_{\Delta_t}(\Delta_t = 0)\, \hat{V}(\mathbf{s}', \mathbf{g}))\right)^2\right], \tag{2.19}$$

where the expectation is taken with respect to $(\mathbf{s}, \mathbf{a}, \mathbf{g}, \mathbf{s}')$ sampled from a replay buffer, $\mathcal{D}$, of data collected by a policy. We approximate the $\hat{V}$-function using a target $Q$-function $Q_{\bar{\phi}}$: $\hat{V}(\mathbf{s}', \mathbf{g}) \approx Q_{\bar{\phi}}(\mathbf{s}', \mathbf{a}', \mathbf{g}) - \log \pi(\mathbf{a}' \mid \mathbf{s}'; \mathbf{g})$, where $\mathbf{a}'$ are samples from the amortized variational policy $\pi_\theta(\cdot \mid \mathbf{s}'; \mathbf{g})$ The parameters $\bar{\phi}$ slowly track the parameters of $\phi$ at each time step via the standard update $\bar{\phi} \leftarrow \eta\bar{\phi} + (1 - \eta)\phi$ [138]. We then train the policy to maximize the approximate $Q$-function by performing gradient descent on

$$\mathcal{F}_\pi(\theta) = -\mathbb{E}_{\mathbf{s}\sim\mathcal{D}, \mathbf{a}\sim\pi_\theta(\cdot|\mathbf{s};\mathbf{g})}\left[Q_\phi(\mathbf{s}, \mathbf{a}, \mathbf{g}) - \log \pi_\theta(\mathbf{a} \mid \mathbf{s}; \mathbf{g})\right]. \tag{2.20}$$

We estimate $\hat{q}_{\Delta_{t+1}}(\Delta_{t+1} = 0)$ with a Monte Carlo estimate of Equation (2.15) obtained via a single Monte Carlo sample $(\mathbf{s}, \mathbf{a}, \mathbf{s}', \mathbf{a}', \mathbf{g})$ from the replay buffer. In practice, a value of



**(a)** Box 2D **(b)** Ant **(c)** Sawyer Push

**(d)** Fetch Push **(e)** Faucet **(f)** Window

**Figure 2.3:** From left to right, we evaluate on: a 2D environment in which an agent must move around a box, a locomotion task in which a quadruped robot must match a location and pose (yellow), and four manipulation tasks in which the robot must push objects, rotate faucet valve, or open a window.

$q_{\Delta_{t+1}}(\Delta_{t+1} = 0) = 1$ can lead to numerical instabilities with bootstrapping, and so we also upper bound the estimated $q_{\Delta_{t+1}}(\Delta_{t+1} = 0)$ by the prior distribution $p_{\Delta_{t+1}}(\Delta_{t+1} = 0)$.

To compute the rewards, we need to compute the likelihood of achieving the desired outcome. If the transition dynamics are unknown, we learn a dynamics model from environment interactions by training a neural network $p_\psi$ that parameterizes the mean and scale of a factorized Laplace distribution. We train this model by maximizing the log-likelihood of the data collected by the policy,

$$\mathcal{F}_p(\psi) = \mathbb{E}_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim\mathcal{D}}[\log p_\psi(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})], \tag{2.21}$$

and use it to compute the rewards

$$\hat{r}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) \doteq \hat{q}_{\Delta_{t+1}}(\Delta_{t+1} = 1) \log p_\psi(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) - D_{\mathrm{KL}}\left(q_{\Delta_t} \middle|\middle| p_{\Delta_t}\right). \tag{2.22}$$

The complete algorithm is presented in Algorithm 1 and consists of alternating between collecting data via policy $\pi$ and minimizing Equations 2.19, 2.20, and 2.21 via gradient descent. This algorithm alternates between approximating the lower bound in Equation (2.12) by repeatedly applying the outcome-driven Bellman operator to an approximate $Q$-function, and maximizing this lower bound by performing approximate policy optimization on Equation (2.20).

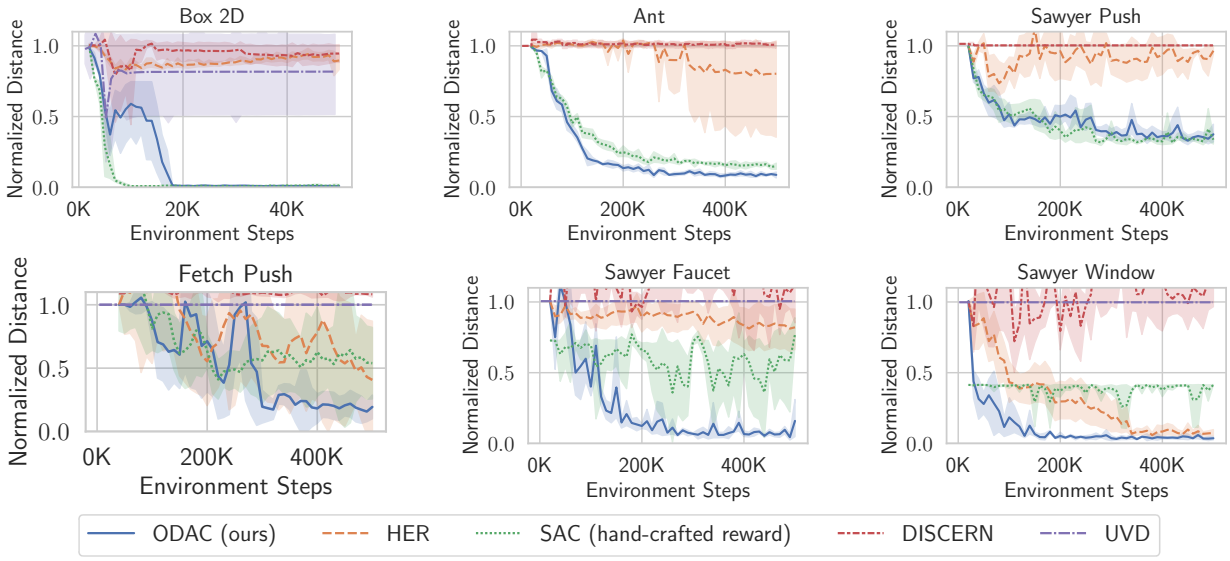



**Figure 2.4:** Learning curves showing final distance vs environment steps across all six environments. We see that only ODAC consistently performs well on all six tasks. Prior methods struggle to learn, especially in the absence of uniform goal sampling. See text for details.

### 2.3.5 Empirical Evaluation

Our experiments compare ODAC to prior methods for learning goal-conditioned policies and evaluate how the components of our variational objective impact the performance of the method. Specifically, we compare ODAC to prior methods on a wide range of manipulation and locomotion tasks that require achieving a desired outcome. To answer the second question, we conduct several ablation studies and visualize the behavior of $q_{\Delta_{t+1}}(\Delta_{t+1} = 0)$. In our experiments, we use a uniform action prior $p(\mathbf{a})$ and the time prior $p_T$ is geometric with parameter 0.01, i.e., $p_{\Delta_{t+1}}(\Delta_{t+1} = 0) = 0.99$. We begin by describing prior methods and environments used for the experiments.

**Baselines and prior work.** We compare our method to hindsight experience replay (HER) [4], a goal-conditioned method, where the learner receives a reward of $-1$ if it is within an $\epsilon$ distance from the goal, and 0 otherwise, universal value density estimation (UVD) [194], which also uses sparse rewards as well as a generative model of the future occupancy measure to estimate the $Q$-values, and DISCERN [217], which learns a reward function by training a discriminator and using a clipped log-likelihood as the reward. Lastly, we include an oracle Soft Actor–Critic (SAC) baseline that uses a manually designed reward. For the MetaWorld tasks, this baseline uses the benchmark reward for each task. For the remaining environments, this baseline uses the Euclidean distance between the agent's current and the desired outcome for the reward.

**Environments.** To avoid over-fitting to any one setting, we compare to these methods across several different robot morphologies and tasks, all illustrated in Figure 2.3 of the

appendix. We compare ODAC to prior work on a simple 2D navigation task, in which an agent must take non-greedy actions to move around a box, as well as the *Ant*, *Sawyer Push*, and *Fetch Push* simulated robot domains, which have each been studied in prior work on reinforcement learning for reaching goals [4, 152, 155, 173, 194]. For the Ant and Sawyer tasks, desired outcomes correspond to full states (i.e., desired positions and joints). For the Fetch task, we use the same goal representation as in prior work [4] and only represent $\mathbf{g}$ with the position of the object. Lastly, we demonstrate the feasibility of replacing manually designed rewards with our outcome-driven paradigm by evaluating the methods on the *Sawyer Window* and *Sawyer Faucet* tasks from the MetaWorld benchmark [227]. These tasks come with manually designed reward functions, which we replace by simply specify a desired outcome $\mathbf{g}$. We plot the mean and standard deviation of the final Euclidean distance to the desired outcome across four random seeds. We normalize the distance to be 1 at the start of training. For further details, see Appendix B.4.1.

In all tasks, a fixed desired final goal is commanded as the exploration goal on each episode, and during training, the goals are relabeled using the future-style relabeling scheme from Andrychowicz et al. [4]. To challenge the methods, we choose the desired goal to be far from the starting state.

**Results.** In Figure 2.4, we see that ODAC outperforms virtually every method on all tasks, consistently learning faster and often reaching a final distance that is orders of magnitude closer to the desired outcome. The only exception is that the hand-crafted reward learns slightly faster on the 2D task, but this gap is closed within a few ten thousand steps.

#### 2.3.5.1 Ablation Studies and Visualizations

Next, we study the importance of the dynamic discount factor $q_{\Delta_{t+1}}(\Delta_{t+1} = 0)$ and the sensitivity of our method to the dynamics model. On all tasks, we evaluate the performance when the posterior exactly matches the prior, that is, $q_{\Delta_{t+1}}(\Delta_{t+1} = 0) = 0.99$ (labeled **fixed** $q_T$ in Table 2.1). Our analysis in Appendix B.1.3 suggests that this setting is sub-optimal, and this ablation empirically evaluate its benefits. We also measure how the algorithm's performance depends on the accuracy of the learned dynamics model used for the reward in ODAC. To do this, we evaluate ODAC with the dynamics model fixed to a multivariate Laplace distribution with a fixed variance, centered at the previous state (labeled **fixed** $\hat{p}_d$

**Table 2.1:** Ablation results, showing mean final normalized distance ($\times 100$) at the end of training across 4 seeds. Best mean is in bold and standard error in parentheses. ODAC is not sensitive to the dynamics models $\hat{p}_d$ but benefits from the dynamic $q_T$ variant.

| Env | ODAC | fix $\hat{p}_d$ | fix $q_T$ | fix $q_T$, $\hat{p}_d$ |
|---|---|---|---|---|
| 2D | 1.7 (1.2) | 1.2 (.14) | **1.0** (.24) | 1.3 (.29) |
| Ant | **9.4** (.48) | 11 (.57) | 12 (.41) | 13 (.20) |
| Push | 35 (2.7) | **34** (1.5) | 37 (1.5) | 38 (3.1) |
| Fetch | 19 (5.5) | **15** (2.5) | 53 (13) | 66 (15) |
| Window | 5.4 (.62) | **5.0** (.62) | 7.9 (.71) | 6.0 (.12) |
| Faucet | **13** (4.2) | 1 5 (3.3) | 37 (8.3) | 38 (7.2) |

in Table 2.1). This ablation represents an extremely crude model, and good performance with such a model would indicate that our method does not depend on obtaining an particularly accurate model.

In Table 2.1, we see that fixing the distribution $q_T$ deteriorates performance, and that using a learned or fixed model both perform relatively well. These results suggest that the derived optimal variational distribution $q^\star_{\Delta_{t+1}}(\Delta_{t+1} = 0)$ given in Proposition 2 is better not only in theory but also in practice, and that ODAC is not sensitive to the accuracy of the dynamics model.

In Appendix B.3, we present the ablation's full learning curves and additional experiments. For example, we compare ODAC to a variant in which we use the learned dynamics model for model-based planning. We find that using the dynamics model only to compute rewards significantly outperformed the variant where it is used for both computing rewards and model-based planning. This result suggests that ODAC does not require learning a dynamics model that is accurate enough for planning, and that the derived Bellman updates are sufficient for obtaining policies that can achieve desired outcomes. In Figure B.3, we also visualize $q_T$ and find that as the policy reaches an irrecoverable state, $q_{\Delta_{t+1}}(\Delta_{t+1} = 0)$ drops in value, suggesting that ODAC *automatically* learns a dynamic discount factor that terminates an episode when an irrecoverable state is reached.

## 2.4 Conclusion

In this chapter, we presented background on goal-conditioned reinforcement learning. We also presented a probabilistic approach for reaching a desired goal or "outcomes," in settings where no reward function and no termination condition are given. We showed that by framing the problem of achieving desired outcomes as variational inference, we can derive an off-policy RL algorithm, a reward function learnable from environment interactions, and a novel Bellman backup that contains a state–action dependent dynamic discount factor for the reward and bootstrap term. Our experimental results further support that the resulting algorithm can lead to efficient outcome-driven approaches to RL.

The standard goal-conditioned formulation and our probabilistic approach both assume that a goal was provided to an agent in the problem statement. In practice, this goal must come from a user, which can be expensive to provide during training. We discuss in the next chapter how an agent can generate its own goals to enable an agent to autonomously learn goal-reaching behaviors without external supervision.

# Chapter 3

# Generating Goals for Autonomous Practice

Reinforcement learning (RL) algorithms hold the promise of allowing autonomous agents, such as robots, to learn to accomplish arbitrary tasks. However, the standard RL framework involves learning policies that are specific to individual tasks, which are defined by hand-specified reward functions. Agents that exist persistently in the world can prepare to solve diverse tasks by setting their own goals, practicing complex behaviors, and learning about the world around them. In fact, humans are very proficient at setting abstract goals for themselves, and evidence shows that this behavior is already present from early infancy [197], albeit with simple goals such as reaching. The behavior and representation of goals grows more complex over time as they learn how to manipulate objects and locomote. How can we begin to devise a reinforcement learning system that sets its own goals and learns from experience with minimal outside intervention and manual engineering?

In this chapter, we take a step toward this goal by designing an RL framework that jointly learns representations of raw sensory inputs and policies that achieve arbitrary goals under this representation by practicing to reach self-specified random goals during training. We then discuss how to select these random goals to most effectively prepare an agent for downstream tasks.

## 3.1 Reinforcement Learning with Imagined Goals

To provide for automated and flexible goal-setting, we must first choose how a general goal can be specified for an agent interacting with a complex and highly variable environment. Even providing the state of such an environment to a policy is a challenge. For instance, a task that requires a robot to manipulate various objects would require a combinatorial representation, reflecting variability in the number and type of objects in the current scene. Directly using raw sensory signals, such as images, avoids this challenge, but learning from raw images is substantially harder. In particular, pixel-wise Euclidean distance is not an

effective reward function for visual tasks since distances between images do not correspond to meaningful distances between states [174, 230]. Furthermore, although end-to-end model-free reinforcement learning can handle image observations, this comes at a high cost in sample complexity, making it difficult to use in the real world.

We propose to address both challenges by incorporating unsupervised representation learning into goal-conditioned policies. In our method, which is illustrated in Figure 3.1, a representation of raw sensory inputs is learned by means of a latent variable model, which in our case is based on the variational autoencoder (VAE) [116]. This model serves three complementary purposes. First, it provides a more structured representation of sensory inputs for RL, making it feasible to learn from images even in the real world. Second, it allows for sampling of new states, which can be used to set synthetic goals during training to allow the goal-conditioned policy to practice diverse behaviors. We can also more efficiently utilize samples from the environment by relabeling synthetic goals in an off-policy RL algorithm, which makes our algorithm substantially more efficient. Third, the learned representation provides a space where distances are more meaningful than the original space of observations, and can therefore provide well-shaped reward functions for RL. By learning to reach random goals sampled from the latent variable model, the goal-conditioned policy learns about the world and can be used to achieve new, user-specified goals at test-time.

This chapter presents a framework for learning general-purpose goal-conditioned policies that can achieve goals specified with target observations. We call our method reinforcement learning with imagined goals (RIG). RIG combines sample-efficient off-policy goal-conditioned reinforcement learning with unsupervised representation learning. We use representation learning to acquire a latent distribution that can be used to sample goals for unsupervised practice and data augmentation, to provide a well-shaped distance function for reinforcement learning, and to provide a more structured representation for the value function and policy. While several prior methods, discussed in the following section, have sought to learn goal-conditioned policies, we can do so with image goals and observations without a manually specified reward signal. Our experimental evaluation illustrates that our method substantially improves the performance of image-based reinforcement learning, can effectively learn policies for complex image-based tasks, and can be used to learn real-world robotic manipulation skills with raw image inputs. Videos of our method in simulated and real-world environments can be found at `https://sites.google.com/site/visualrlwithimaginedgoals/`.

### 3.1.1 Goal-Conditioned Policies with Unsupervised Representation Learning

To devise a practical algorithm based on goal-conditioned value functions, we must choose a suitable goal representation. In the absence of domain knowledge and instrumentation, a general-purpose choice is to set the goal space $\mathcal{G}$ to be the same as the state observations space $\mathcal{S}$. This choice is fully general as it can be applied to any task, and still permits considerable user control since the user can choose a "goal state" to set a desired goal for a

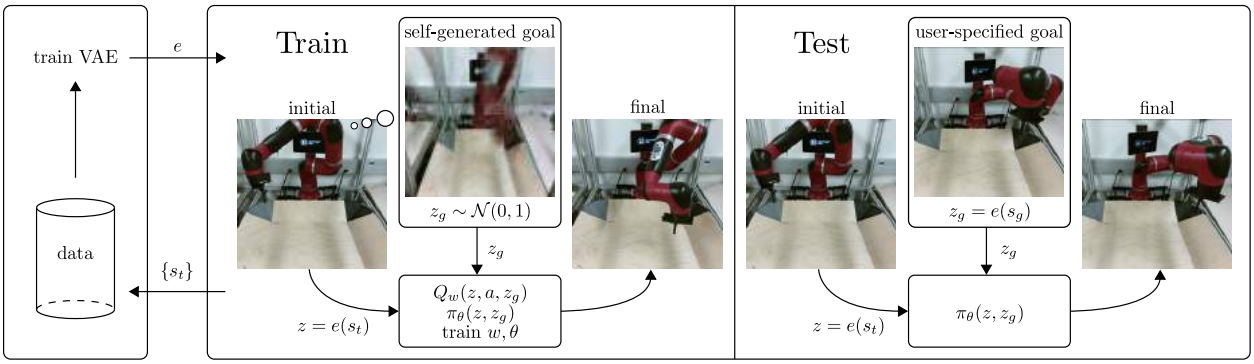



**Figure 3.1:** We train a VAE using data generated by our exploration policy (left). We use the VAE for multiple purposes during training time (middle): to sample goals to train the policy, to embed the observations into a latent space, and to compute distances in the latent space. During test time (right), we embed a specified goal observation $o_g$ into a goal latent $\mathbf{z}_g$ as input to the policy. Videos of our method can be found at `sites.google.com/site/visualrlwithimaginedgoals`

trained goal-conditioned policy. But when the state space $\mathcal{S}$ corresponds to high-dimensional sensory inputs such as images [1] learning a goal-conditioned $Q$-function and policy becomes exceedingly difficult as we illustrate empirically in Section 3.1.2.

Our method jointly addresses a number of problems that arise when working with high-dimensional inputs such as images: sample efficient learning, reward specification, and automated goal-setting. We address these problems by learning a latent embedding using a $\beta$-VAE. We use this latent space to represent the goal and state and retroactively relabel data with latent goals sampled from the VAE prior to improve sample efficiency. We also show that distances in the latent space give us a well-shaped reward function for images. Lastly, we sample from the prior to allow an agent to set and "practice" reaching its own goal, removing the need for humans to specify new goals during training time. We next describe the specific components of our method, and summarize our complete algorithm in Section 3.1.1.5.

#### 3.1.1.1 Sample-Efficient RL with Learned Representations

One challenging problem with end-to-end approaches for visual RL tasks is that the resulting policy needs to learn both perception and control. Rather than operating directly on observations, we embed the state $\mathbf{s}_t$ and goals $\mathbf{g}$ into a latent space $\mathcal{Z}$. To learn a representation of the state and goal space, we train a $\beta$-VAE by executing a random policy and collecting state observations, $\{\mathbf{s}^{(i)}\}$, and optimize Equation (2.5). We then use the mean of the encoder as the state encoding, i.e. $\mathbf{z} = e(\mathbf{s}) \doteq \mu_\phi(\mathbf{s})$.

After training the VAE, we train a goal-conditioned $Q$-function $Q(\mathbf{z}, \mathbf{a}, \mathbf{z}_g)$ and corresponding policy $\pi_\theta(\mathbf{z}, \mathbf{z}_g)$ in this latent space. The policy is trained to reach a goal $\mathbf{z}_g$ using

[1] We make the simplifying assumption that the system is Markovian with respect to the sensory input. One could incorporate memory into the state for partially observed tasks.

the reward function discussed in Section 3.1.1.2. For the underlying RL algorithm, we use twin delayed deep deterministic policy gradients (TD3) [71], though any value-based RL algorithm could be used. Note that the policy (and $Q$-function) operates completely in the latent space. During test time, to reach a specific goal state $\mathbf{g}$, we encode the goal $\mathbf{z}_g = e(\mathbf{g})$ and input this latent goal to the policy.

As the policy improves, it may visit parts of the state space that the VAE was never trained on, resulting in arbitrary encodings that may not make learning easier. Therefore, in addition to procedure described above, we fine-tune the VAE using both the randomly generated state observations $\{s^{(i)}\}$ and the state observations collected during exploration. We show in **??** C.1.1.3 that this additional training helps the performance of the algorithm.

#### 3.1.1.2 Reward Specification

Training the goal-conditioned value function requires defining a goal-conditioned reward $r(\mathbf{s}, \mathbf{g})$. Using Euclidean distances in the space of image pixels provides a poor metric, since similar configurations in the world can be extremely different in image space. In addition to compactly representing high-dimensional observations, we can utilize our representation to obtain a reward function based on the reward function derived in Section 2.3.

That reward function is the negative Mahalanobis distance in the latent space:

$$r(\mathbf{s}, \mathbf{g}) = -||e(\mathbf{s}) - e(\mathbf{g})||_A = -||\mathbf{z} - \mathbf{z}_g||_A,$$

where the matrix $A$ weights different dimensions in the latent space. To see the relationship to our previously derived reward, let $A$ to be the precision matrix of the VAE encoder, $q_\phi$. Since we use a Gaussian encoder, we have that

$$r(\mathbf{s}, \mathbf{g}) = -||\mathbf{z} - \mathbf{z}_g||_A \propto \sqrt{\log q_\phi(\mathbf{z}_g \mid \mathbf{s})}. \tag{3.1}$$

In other words, minimizing this squared distance in the latent space is equivalent to rewarding reaching states that maximize the probability of the latent goal $\mathbf{z}_g$, which approximates the reward derived in Equation (2.14) when we a fix the posterior over when we will reach the goal to the prior, i.e., $q_{\Delta_t} = p_{\Delta_t}$.

In practice, we found that setting $A = \mathbf{I}$, corresponding to Euclidean distance, performed better than Mahalanobis distance, though its effect is the same — to bring $\mathbf{z}$ close to $\mathbf{z}_g$ and maximize the probability of the latent goal $\mathbf{z}_g$ given the observation. This interpretation would not be possible when using normal autoencoders since distances are not trained to have any probabilistic meaning. Indeed, we show in Section 3.1.2 that using distances in a normal autoencoder representation often does not result in meaningful behavior.

#### 3.1.1.3 Improving Sample Efficiency with Latent Goal Relabeling

To further enable sample-efficient learning in the real world, we use the VAE to relabel goals. Note that we can optimize Equation (2.2) using any valid $(\mathbf{s}, \mathbf{a}, \mathbf{s}', \mathbf{g}, r)$ tuple. If we

could artificially generate these tuples, then we could train our entire RL algorithm without collecting any data. Unfortunately, we do not know the system dynamics, and therefore have to sample transitions $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ by interacting with the world. However, we have the freedom to relabel the goal and reward synthetically. So if we have a mechanism for generating goals and computing rewards, then given $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$, we can generate a new goal $\mathbf{g}$ and new reward $r(\mathbf{s}, \mathbf{a}, \mathbf{s}', \mathbf{g})$ to produce a new tuple $(\mathbf{s}, \mathbf{a}, \mathbf{s}', \mathbf{g}, r)$. By artificially generating and recomputing rewards, we can convert a single $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ transition into potentially infinitely many valid training datums.

For image-based tasks, this procedure would require generating goal images, an onerous task on its own. However, our reinforcement learning algorithm operates directly in the latent space for goals and rewards. So rather than generating goals $\mathbf{g}$, we generate latent goals $\mathbf{z}_g$ by sampling from the VAE prior $p(\mathbf{z})$. We then recompute rewards using Equation (3.1). By retroactively relabeling the goals and rewards, we obtain much more data to train our value function. This sampling procedure is made possible by our use of a latent variable model, which is explicitly trained so that sampling from the latent distribution is straightforward.

In practice, the distribution of latents will not exactly match the prior. To mitigate this distribution mismatch, we use a fitted prior when sampling from the prior: we fit a diagonal Gaussian to the latent encodings of the VAE training data, and use this fitted prior in place of the unit Gaussian prior.

Retroactively generating goals is also explored in tabular domains by Kaelbling [104] and in continuous domains by Andrychowicz et al. [4] using hindsight experience replay (HER). However, HER is limited to sampling goals seen along a trajectory, which greatly limits the number and diversity of goals with which one can relabel a given transition. Our final method uses a mixture of the two strategies: half of the goals are generated from the prior and half from goals use the "future" strategy described in Andrychowicz et al. [4]. We show in Section 3.1.2 that relabeling the goal with samples from the VAE prior results in significantly better sample-efficiency.

#### 3.1.1.4 Automated Goal-Generation for Exploration

If we do not know which particular goals will be provided at test time, we would like our RL agent to carry out a self-supervised "practice" phase during training, where the algorithm proposes its own goals, and then practices how to reach them. Since the VAE prior represents a distribution over latent goals and state observations, we again sample from this distribution to obtain plausible goals. After sampling a goal latent from the prior $\mathbf{z}_g \sim p(\mathbf{z})$, we give this to our policy $\pi(z, \mathbf{z}_g)$ to collect data.

#### 3.1.1.5 Algorithm Summary

We call the complete algorithm reinforcement learning with imagined goals (RIG) and summarize it in Algorithm 2. We first collect data with a simple exploration policy, though any exploration strategy could be used for this stage, including off-the-shelf exploration

**Algorithm 2** RIG: Reinforcement learning with imagined goals

**Require:** VAE encoder $q_\phi$, VAE decoder $p_\theta$, policy $\pi_\theta$, goal-conditioned value function $Q_w$.
Collect $\mathcal{D} = \{\mathbf{s}^{(i)}\}$ using exploration policy.
Train $\beta$-VAE on $\mathcal{D}$ by optimizing (2.5).
Fit prior $p(\mathbf{z})$ to latent encodings $\{\mu_\phi(\mathbf{s}^{(i)})\}$.
**for** $n = 0, ..., N-1$ episodes **do**
  Sample latent goal from prior $\mathbf{z}_g \sim p(\mathbf{z})$.
  Sample initial state $\mathbf{s}_0 \sim E$.
  **for** $t = 0, ..., H-1$ steps **do**
    Get action $\mathbf{a}_t = \pi_\theta(e(\mathbf{s}_t), \mathbf{z}_g) +$ noise.
    Get next state $\mathbf{s}_{t+1} \sim p(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)$.
    Store $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{z}_g)$ into replay buffer $\mathcal{R}$.
    Sample transition $(\mathbf{s}, \mathbf{a}, \mathbf{s}', \mathbf{z}_g) \sim \mathcal{R}$.
    Encode $z = e(\mathbf{s}), z' = e(\mathbf{s}')$.
    (Probability 0.5) replace $\mathbf{z}_g$ with $\mathbf{z}'_g \sim p(\mathbf{z})$.
    Compute new reward $r = -||z' - \mathbf{z}_g||$.
    Minimize (2.2) using $(z, \mathbf{a}, z', \mathbf{z}_g, r)$.
  **end for**
  **for** $t = 0, ..., H-1$ steps **do**
    **for** $i = 0, ..., k-1$ steps **do**
      Sample future state $\mathbf{s}_{h_i}$, $t < h_i \leq H-1$.
      Store $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, e(\mathbf{s}_{h_i}))$ into $\mathcal{R}$.
    **end for**
  **end for**
  Fine-tune $\beta$-VAE every $K$ episodes on mixture of $\mathcal{D}$ and $\mathcal{R}$.
**end for**

bonuses [166, 13] or unsupervised reinforcement learning methods [58, 66]. Then, we train a VAE latent variable model on state observations and finetune it over the course of training. We use this latent variable model for multiple purposes: We sample a latent goal $\mathbf{z}_g$ from the model and condition the policy on this goal. We embed all states and goals using the model's encoder. When we train our goal-conditioned value function, we resample goals from the prior and compute rewards in the latent space using Equation (3.1). Any RL algorithm that trains $Q$-functions could be used, and we use TD3 [71] in our implementation.

### 3.1.2 Experiments

Our experiments address the following questions:

1. How does our method compare to prior model-free RL algorithms in terms of sample

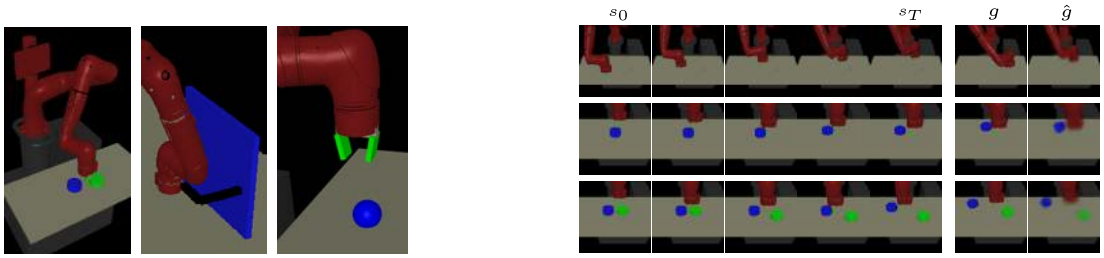



**Figure 3.2:** (Left) The simulated pusher, door opening, and pick-and-place environments are pictured. (Right) Test rollouts from our learned policy on the three pushing environments. Each row is one rollout. The right two columns show a goal image $\mathbf{g}$ and its VAE reconstruction $\hat{\mathbf{g}}$. The images to their left show frames from a trajectory to reach the given goal.

efficiency and performance, when learning continuous control tasks from images?

2. How critical is each component of our algorithm for efficient learning?

3. Does our method work on tasks where the state space cannot be easily specified ahead of time, such as tasks that require interaction with variable numbers of objects?

4. Can our method scale to real world vision-based robotic control tasks?

For the first two questions, we evaluate our method against a number of prior algorithms and ablated versions of our approach on a suite of the following simulated tasks. *Visual Reacher*: a MuJoCo [206] environment with a 7-dof Sawyer arm reaching goal positions. The arm is shown the left of Figure 3.2. The end-effector (EE) is constrained to a 2-dimensional rectangle parallel to a table. The action controls EE velocity within a maximum velocity. *Visual Pusher*: a MuJoCo environment with a 7-dof Sawyer arm and a small puck on a table that the arm must push to a target push. *Visual Multi-Object Pusher*: a copy of the Visual Pusher environment with two pucks. *Visual Door*: a Sawyer arm with a door it can attempt to open by latching onto the handle. *Visual Pick and Place*: a Sawyer arm with a small ball and an additional dimension of control for opening and closing the gripper. Detailed descriptions of the environments are provided in the Supplementary Material.

Solving these tasks directly from images poses a challenge since the controller must learn both perception and control. The evaluation metric is the distance of objects (including the arm) to their respective goals. To evaluate our policy, we set the environment to a sampled goal position, capture an image, and encode the image to use as the goal. Although we use the ground-truth positions for evaluation, **we do not use the ground-truth positions for training the policies**. The only inputs from the environment that our algorithm receives are the image observations. For Visual Reacher, we pretrained the VAE with 100 images. For other tasks, we used 10,000 images.

We compare our method with the following prior works. *L&R*: Lange and Riedmiller [125] trains an autoencoder to handle images. *DSAE*: Deep spatial autoencoders [64] learns a spatial autoencoder and uses guided policy search [134] to achieve a single goal image. *HER*: Hindsight experience replay [4] utilizes a sparse reward signal and relabeling trajectories with achieved goals. *Oracle*: RL with direct access to state information for observations and rewards.

To our knowledge, no prior work demonstrates policies that can reach a variety of goal images without access to a true-state reward function, and so we needed to make modifications to make the comparisons feasible. L&R assumes a reward function from the environment. Since we have no state-based reward function, we specify the reward function as distance in the autoencoder latent space. HER does not embed inputs into a latent space but instead operates directly on the input, so we use pixel-wise mean squared error (MSE) as the metric. DSAE is trained only for a single goal, so we allow the method to generalize to a variety of test goal images by using a goal-conditioned $Q$-function. To make the implementations

[2]In all our simulation results, each plot shows a 95% confidence interval of the mean across 5 seeds.

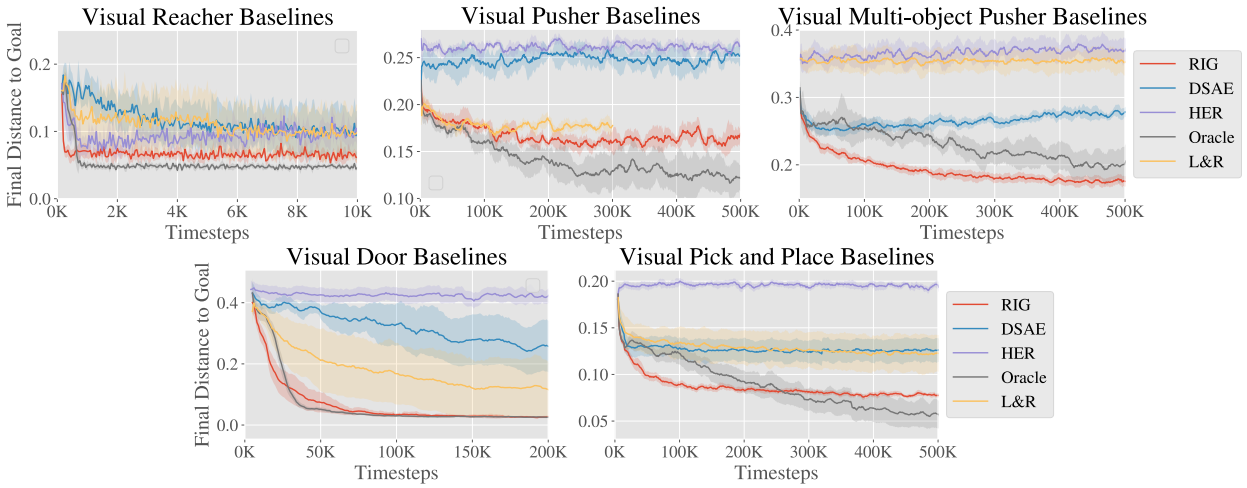



**Figure 3.3:** Simulation results, final distance to goal vs simulation steps[2]. RIG (red) consistently outperforms the baselines, except for the oracle which uses ground truth object state for observations and rewards. On the hardest tasks, only our method and the oracle discover viable solutions.

comparable, we use the same off-policy algorithm, TD3 [71], to train L&R, HER, and our method. Unlike our method, prior methods do not specify how to select goals during training, so we favorably give them real images as goals for rollouts, sampled from the same distribution that we use to test.

We see in Figure 3.3 that our method can efficiently learn policies from visual inputs to perform simulated reaching and pushing, without access to the object state. Our approach substantially outperforms the prior methods, for which the use of image goals and observations poses a major challenge. HER struggles because pixel-wise MSE is hard to optimize. Our latent-space rewards are much better shaped and allow us to learn more complex tasks. Finally, our method is close to the state-based "oracle" method in terms of sample efficiency and performance, without having any access to object state. Notably, in the multi-object environment, our method actually outperforms the oracle, likely because the state-based reward contains local minima. Overall, these result show that our method is capable of handling raw image observations much more effectively than previously proposed goal-conditioned RL methods. Next, we perform ablations to evaluate our contributions in isolation. Results on Visual Pusher are shown, and see Appendix C.1.1 for experiments on all three simulated environments.

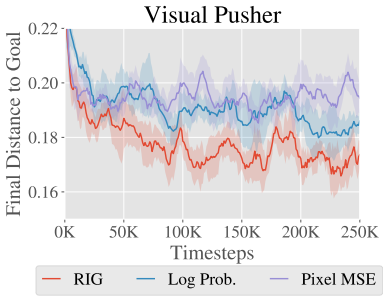



**Figure 3.4:** Reward type ablation results. RIG (red), which uses latent Euclidean distance, outperforms the other methods.

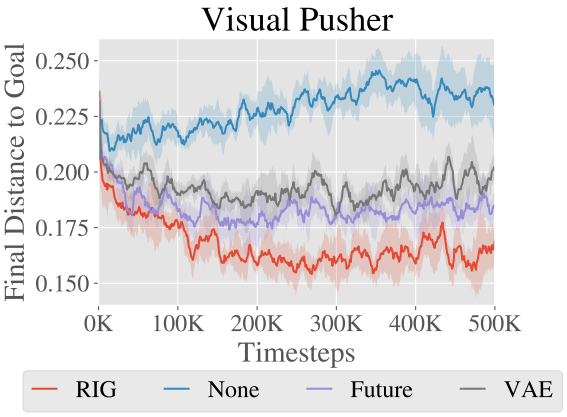



**Figure 3.5:** Training curve for relabeling ablation. We see that the RIG-style relabeling that relabels with both future and self-generated goals performs the best.

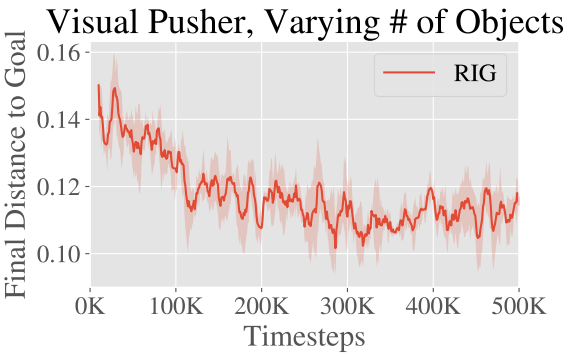



**Figure 3.6:** Training curve for learning with varying number of objects. We see that RIG makes progress on this task, which does not length itself to a fix-length state representation.

**Reward Specification Comparison** We evaluate how effective distance in the VAE latent space is for the Visual Pusher task. We keep our method the same, and only change the reward function that we use to train the goal-conditioned valued function. We include the following methods for comparison: *Latent Distance*, which uses the reward used in RIG, i.e. $A = \mathbf{I}$ in Equation (3.1); *Log Probability*, which uses the Mahalanobis distance in Equation (3.1), where $A$ is the precision matrix of the encoder; and *Pixel MSE*, which uses mean-squared error (MSE) between state and goal in pixel space. [3] In Figure 3.4, we see that latent distance significantly outperforms log probability. We suspect that small variances of the VAE encoder results in drastically large rewards, making the learning more difficult. We also see that latent distances results in faster learning when compared to pixel MSE.

**Relabeling Strategy Comparison** As described in Section 3.1.1.3, our method uses a novel goal relabeling method based on sampling from the generative model. To isolate how much our new goal relabeling method contributes to our algorithm, we vary the resampling strategy while fixing other components of our algorithm. The resampling strategies that we consider are: *Future*, relabeling the goal for a transition by sampling uniformly from future states in the trajectory as done in Andrychowicz et al. [4]; *VAE*, sampling goals from the VAE only; *RIG*, relabeling goals with probability 0.5 from the VAE and probability 0.5 using the future strategy; and *None*, no relabeling. In Figure 3.5, we see that sampling from the VAE and Future is significantly better than not relabeling at all. In RIG, we use an equal mixture of the VAE and Future sampling strategies, which performs best by a large margin. **??** C.1.1.1 contains results on all simulated environments, and **??** C.1.1.4 considers relabeling strategies with a known goal distribution.

[3] To compute the pixel MSE for a sampled latent goal, we decode the goal latent using the VAE decoder, $p_\psi$, to generate the corresponding goal image.

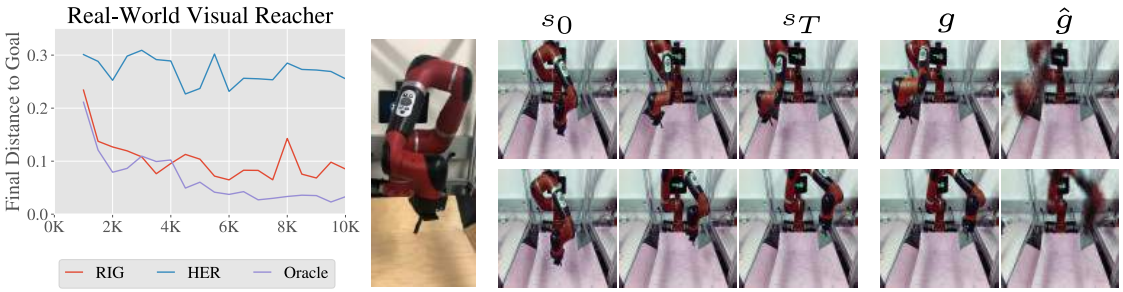
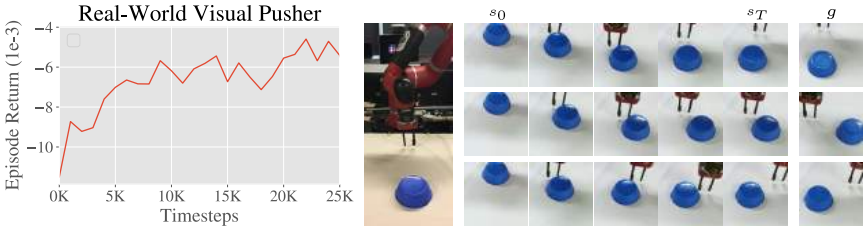



**Figure 3.7:** (Left) Our method compared to the HER baseline and oracle on a real-world visual reaching task. (Middle) Our robot setup is pictured. (Right) Test rollouts of our learned policy.




Puck Distance to Goal (cm)

| RIG | HER |
|---|---|
| $\mathbf{4.5 \pm 2.5}$ | $14.9 \pm 5.4$ |

**Figure 3.8:** (Left) The learning curve for real-world pushing. (Middle) Our robot pushing setup is pictured, with frames from test rollouts of our learned policy. (Right) Our method compared to the HER baseline on the real-world visual pushing task. We evaluated the performance of each method by manually measuring the distance between the goal position of the puck and final position of the puck for 15 test rollouts, reporting mean and standard deviation.

**Learning with Variable Numbers of Objects** A major advantage of working directly from pixels is that the policy input can easily represent combinatorial structure in the environment, which would be difficult to encode into a fixed-length state vector even if a perfect perception system were available. For example, if a robot has to interact with different combinations and numbers of objects, picking a single MDP state representation would be challenging, even with access to object poses. By directly processing images for both the state and the goal, no modification is needed to handle the combinatorial structure: the number of pixels always remains the same, regardless of how many objects are in the scene.

We demonstrate that our method can handle this difficult scenario by evaluating on a task where the environment, based on the Visual Multi-Object Pusher, randomly contains zero, one, or two objects in each episode during testing. During training, each episode still always starts with both objects in the scene, so the experiments tests whether a trained policy can handle variable numbers of objects at test time. Figure 3.6 shows that our method can learn to solve this task successfully, without decrease in performance from the base setting where both objects are present (in Figure 3.3). Developing and demonstrating algorithms that solve tasks with varied underlying structure is an important step toward creating autonomous agents that can handle the diversity of tasks present "in the wild."

#### 3.1.2.1 Visual RL with Physical Robots

RIG is a practical and straightforward algorithm to apply to real physical systems: the efficiency of off-policy learning with goal relabeling makes training times manageable, while the use of image-based rewards through the learned representation frees us from the burden of manually design reward functions, which itself can require hand-engineered perception systems [187]. We trained policies for visual reaching and pushing on a real-world Sawyer robotic arm, shown in Figure 3.7. The control setup matches Visual Reacher and Visual Pusher respectively, meaning that **the only input from the environment consists of camera images**.

We see in Figure 3.7 that our method is applicable to real-world robotic tasks, almost matching the state-based oracle method and far exceeding the baseline method on the reaching task. Our method needs just 10,000 samples or about an hour of real-world interaction time to solve visual reaching.

Real-world pushing results are shown in Figure 3.8. To solve visual pusher, which is more visually complicated and requires reasoning about the contact between the arm and object, our method requires about 25,000 samples, which is still a reasonable amount of real-world training time. Note that unlike previous results, we do not have access to the true puck position during training so for the learning curve we report test episode returns on the VAE latent distance reward. We see RIG making steady progress at optimizing the latent distance as learning proceeds.

These experiments demonstrate that RIG is a promising method for enabling agents to autonomously acquire goal-directed skills. Because RIG uses the generative model to sample new goals for exploration, one limitation of RIG is that the resulting exploration is sensitive to the data used to train the generative model. On more complex domains, a simple exploration strategy as mentioned in Section 3.1.1.5 is unlikely to be sufficient for generative diverse data that will encourage the In other words, although RIG enables an agent to generate new goals for autonomous exploration, it does not guarantee that the generated goals will be interested, nor does it formally define what constitutes "interesting" goals. We address this limitation in the next section.

## 3.2 Skew-Fit: Setting the Right Goals

How do we design an unsupervised RL algorithm that automatically explores the environment and iteratively distills this experience into general-purpose policies that can accomplish new user-specified tasks at test time? In Section 3.1, we presented RIG, an algorithm that enables an RL agent to autonomously generate goals and practice reaching them. At test time, the trained policy can reach some user-specified goals without any additional training, but the policy's test time performance depends on the set of goals that the policy autonomously generates during the unsupervised phase of learning.

In the absence of any prior knowledge, an effective exploration scheme is one that visits

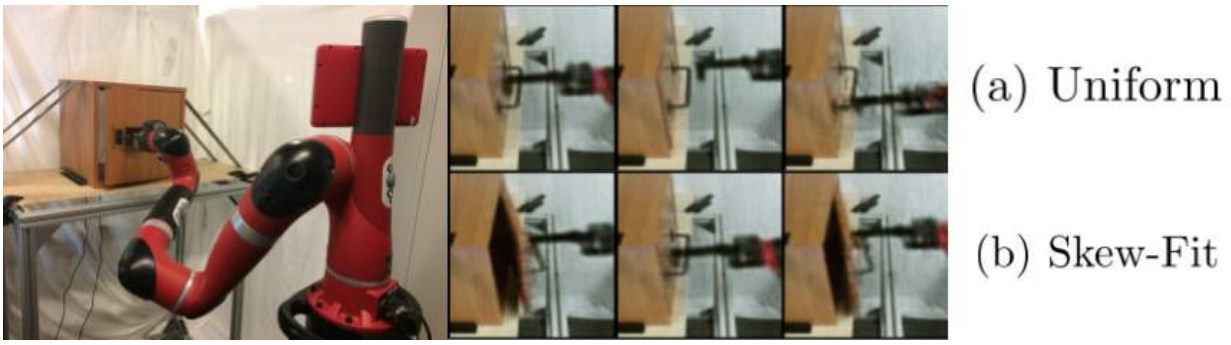



**Figure 3.9:** Left: Robot learning to open a door with Skew-Fit, without any task reward. Right: Samples from a goal distribution when using (a) uniform and (b) Skew-Fit sampling. When used as goals, the diverse samples from Skew-Fit encourage the robot to practice opening the door more frequently.

as many states as possible, allowing a policy to autonomously prepare for user-specified tasks that it might see at test time. We can formalize the problem of visiting as many states as possible as one of maximizing the *state entropy* $\mathcal{H}(\mathbf{s})$ under the current policy.[5] Unfortunately, optimizing this objective alone does not result in a policy that can solve new tasks: it only knows how to maximize state entropy. In other words, to develop principled unsupervised RL algorithms that result in useful policies, maximizing $\mathcal{H}(\mathbf{s})$ is not enough. We need a mechanism that allows us to reuse the resulting policy to achieve new tasks at test-time.

We argue that this can be accomplished by performing *goal-directed exploration*: a policy should autonomously visit as many states as possible, but after autonomous exploration, a user should be able to reuse this policy by giving it a goal $\mathbf{G}$ that corresponds to a state that it must reach. While not all test-time tasks can be expressed as reaching a goal state, a wide range of tasks can be represented in this way. Mathematically, the goal-conditioned policy should minimize the conditional entropy over the states given a goal, $\mathcal{H}(\mathbf{s} \mid \mathbf{G})$, so that there is little uncertainty over its state given a commanded goal. This objective provides us with a principled way to train a policy to explore all states (maximize $\mathcal{H}(\mathbf{s})$) such that the state that is reached can be determined by commanding goals (minimize $\mathcal{H}(\mathbf{s} \mid \mathbf{G})$).

Directly optimizing this objective is in general intractable, since it requires optimizing the entropy of the marginal state distribution, $\mathcal{H}(\mathbf{s})$. However, we can sidestep this issue by noting that the objective is the mutual information between the state and the goal, $I(\mathbf{s}; \mathbf{G})$, which can be written as:

$$\mathcal{H}(\mathbf{s}) - \mathcal{H}(\mathbf{s}|\mathbf{G}) = I(\mathbf{s}; \mathbf{G}) = \mathcal{H}(\mathbf{G}) - \mathcal{H}(\mathbf{G}|\mathbf{s}). \tag{3.2}$$

Equation (3.2) thus gives an equivalent objective for an unsupervised RL algorithm: the agent should set diverse goals, maximizing $\mathcal{H}(\mathbf{G})$, and learn how to reach them, minimizing $\mathcal{H}(\mathbf{G} \mid \mathbf{s})$.

[5] We consider the distribution over terminal states in a finite horizon task and believe this work can be extended to infinite horizon stationary distributions.

While learning to reach goals is the typical objective studied in goal-conditioned RL [104, 4], setting goals that have maximum diversity is crucial for effectively learning to reach all possible states. Acquiring such a maximum-entropy goal distribution is challenging in environments with complex, high-dimensional state spaces, where even knowing which states are valid presents a major challenge. For example, in image-based domains, a uniform goal distribution requires sampling uniformly from the set of realistic images, which in general is unknown a priori.

This chapter presents the following contributions. First, we propose a principled objective for unsupervised RL, based on Equation (3.2). While a number of prior works ignore the $\mathcal{H}(\mathbf{G})$ term, we argue that jointly optimizing the entire quantity is needed to develop effective exploration. Second, we present a general algorithm called Skew-Fit and prove that under regularity conditions Skew-Fit learns a sequence of generative models that converges to a uniform distribution over the goal space, even when the set of valid states is unknown (e.g., as in the case of images). Third, we describe a concrete implementation of Skew-Fit and empirically demonstrate that this method achieves state of the art results compared to a large number of prior methods for goal reaching with visually indicated goals, including a real-world manipulation task, which requires a robot to learn to open a door from scratch in about five hours, directly from images, and without any manually-designed reward function.

### 3.2.1 Problem Formulation

To ensure that an unsupervised reinforcement learning agent learns to reach all possible states in a controllable way, we maximize the mutual information between the state $\mathbf{s}$ and the goal $\mathbf{G}$, $I(\mathbf{s}; \mathbf{G})$, as stated in Equation (3.2). This section discusses how to optimize Equation (3.2) by splitting the optimization into two parts: minimizing $\mathcal{H}(\mathbf{G} \mid \mathbf{s})$ and maximizing $\mathcal{H}(\mathbf{G})$.

#### 3.2.1.1 Minimizing $\mathcal{H}(\mathbf{G} \mid \mathbf{s})$: Goal-Conditioned Reinforcement Learning

For simplicity, in this chappter we will assume in our derivation that the goal space matches the state space, such that $\mathcal{G} = \mathcal{S}$, though the approach extends trivially to the case where $\mathcal{G}$ is a hand-specified subset of $\mathcal{S}$, such as the global XY position of a robot.

Goal-reaching can be formulated as minimizing $\mathcal{H}(\mathbf{G} \mid \mathbf{s})$, and many practical goal-reaching algorithms [104, 138, 190, 4, 155, 172, 67] can be viewed as approximations to this objective by observing that the optimal goal-conditioned policy will deterministically reach the goal, resulting in a conditional entropy of zero: $\mathcal{H}(\mathbf{G} \mid \mathbf{s}) = 0$. See Appendix C.2.5 for more details. Our method may thus be used in conjunction with any of these prior goal-conditioned RL methods in order to jointly minimize $\mathcal{H}(\mathbf{G} \mid \mathbf{s})$ and maximize $\mathcal{H}(\mathbf{G})$.

#### 3.2.1.2 Maximizing $\mathcal{H}(\mathbf{G})$: Setting Diverse Goals

We now turn to the problem of setting diverse goals or, mathematically, maximizing the entropy of the goal distribution $\mathcal{H}(\mathbf{G})$. Let $U_{\mathcal{S}}$ be the uniform distribution over $\mathcal{S}$, where we

assume $\mathcal{S}$ has finite volume so that the uniform distribution is well-defined. Let $q_\phi^G$ be the goal distribution from which goals $\mathbf{G}$ are sampled, parameterized by $\phi$. Our goal is to maximize the entropy of $q_\phi^G$, which we write as $\mathcal{H}(\mathbf{G})$. Since the maximum entropy distribution over $\mathcal{S}$ is the uniform distribution $U_\mathcal{S}$, maximizing $\mathcal{H}(\mathbf{G})$ may seem as simple as choosing the uniform distribution to be our goal distribution: $q_\phi^G = U_\mathcal{S}$. However, this requires knowing the uniform distribution over valid states, which may be difficult to obtain when $\mathcal{S}$ is a subset of $\mathbb{R}^n$, for some $n$. For example, if the states correspond to images viewed through a robot's camera, $\mathcal{S}$ corresponds to the (unknown) set of valid images of the robot's environment, while $\mathbb{R}^n$ corresponds to all possible arrays of pixel values of a particular size. In such environments, sampling from the uniform distribution $\mathbb{R}^n$ is unlikely to correspond to a valid image of the real world. Sampling uniformly from $\mathcal{S}$ would require knowing the set of all possible valid images, which we assume the agent does not know when starting to explore the environment.

While we cannot sample arbitrary states from $\mathcal{S}$, we can sample states by performing goal-directed exploration. To derive and analyze our method, we introduce a simple model of this process: a goal $\mathbf{G} \sim q_\phi^G$ is sampled from the goal distribution $q_\phi^G$, and then the goal-conditioned policy $\pi$ attempts to achieve this goal, which results in a distribution of terminal states $\mathbf{s} \in \mathcal{S}$. We abstract this entire process by writing the resulting marginal distribution over $\mathbf{s}$ as $p_\phi^S(\mathbf{s}) \doteq \int_\mathcal{G} q_\phi^G(\mathbf{G}) p(\mathbf{s} \mid \mathbf{G}) d\mathbf{G}$, where the subscript $\phi$ indicates that the marginal $p_\phi^S$ depends indirectly on $q_\phi^G$ via the goal-conditioned policy $\pi$. We assume that $p_\phi^S$ has full support, which can be accomplished with an epsilon-greedy goal reaching policy in a communicating MDP. We also assume that the entropy of the resulting state distribution $\mathcal{H}(p_\phi^S)$ is no less than the entropy of the goal distribution $\mathcal{H}(q_\phi^G)$. Without this assumption, a policy could ignore the goal and stay in a single state, no matter how diverse and realistic the goals are. [6] This simplified model allows us to analyze the behavior of our goal-setting scheme separately from any specific goal-reaching algorithm. We will however show in Section 3.2.4 that we can instantiate this approach into a practical algorithm that jointly learns the goal-reaching policy. In summary, our goal is to acquire a maximum-entropy goal distribution $q_\phi^G$ over valid states $\mathcal{S}$, while only having access to state samples from $p_\phi^S$.

### 3.2.2 Skew-Fit: Learning a Maximum Entropy Goal Distribution

Our method, Skew-Fit, learns a maximum entropy goal distribution $q_\phi^G$ using samples collected from a goal-conditioned policy. We analyze the algorithm and show that Skew-Fit maximizes the goal distribution entropy, and present a practical instantiation for unsupervised deep RL.

#### 3.2.2.1 Skew-Fit Algorithm

To learn a uniform distribution over *valid* goal states, we present a method that iteratively increases the entropy of a generative model $q_\phi^G$. In particular, given a generative model

[6] Note that this assumption does **not** require that the entropy of $p_\phi^S$ is strictly larger than the entropy of the goal distribution, $q_\phi^G$.

$q^G_{\phi_t}$ at iteration $t$, we want to train a new generative model, $q^G_{\phi_{t+1}}$ that has higher entropy. While we do not know the set of valid states $\mathcal{S}$, we could sample states $\mathbf{s}_n \overset{\text{iid}}{\sim} p^S_{\phi_t}$ using the goal-conditioned policy, and use the samples to train $q^G_{\phi_{t+1}}$. However, there is no guarantee that this would increase the entropy of $q^G_{\phi_{t+1}}$.

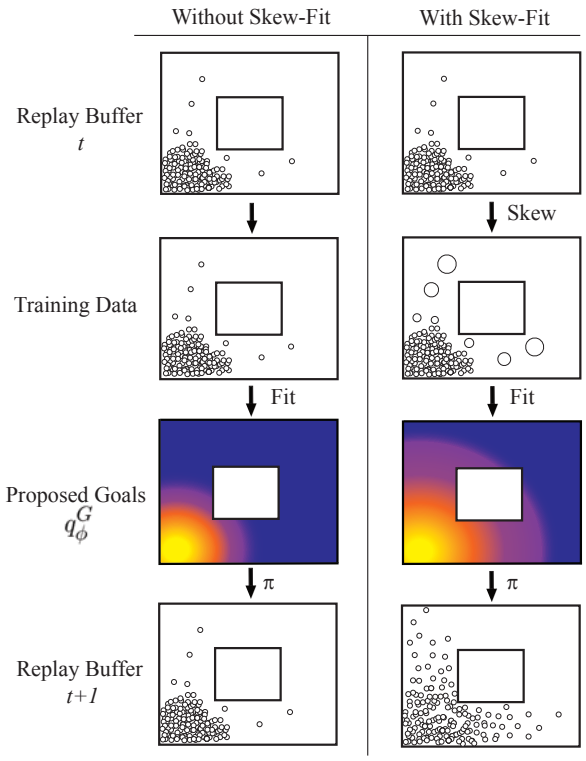



**Figure 3.10:** Our method, Skew-Fit, samples goals for goal-conditioned RL. We sample states from our replay buffer, and give more weight to rare states. We then train a generative model $q^G_{\phi\ t+1}$ with the weighted samples. By sampling new states with goals proposed from this new generative model, we obtain a higher entropy state distribution in the next iteration.

The intuition behind our method is simple: rather than fitting a generative model to these samples $\mathbf{s}_n$, we *skew* the samples so that rarely visited states are given more weight. See Figure 3.10 for a visualization of this process. How should we skew the samples if we want to maximize the entropy of $q^G_{\phi_{t+1}}$? If we had access to the density of each state, $p^S_{\phi_t}(\mathbf{s})$, then we could simply weight each state by $1/p^S_{\phi_t}(\mathbf{s})$. We could then perform maximum likelihood estimation (MLE) for the uniform distribution by using the following importance sampling

(IS) loss to train $\phi_{t+1}$:

$$\begin{aligned}\mathcal{L}(\phi) &= \mathbb{E}_{\mathbf{s}\sim U_{\mathcal{S}}}\left[\log q_\phi^G(\mathbf{s})\right] \\ &= \mathbb{E}_{\mathbf{s}\sim p_{\phi_t}^S}\left[\frac{U_{\mathcal{S}}(\mathbf{s})}{p_{\phi_t}^S(\mathbf{s})}\log q_\phi^G(\mathbf{s})\right] \\ &\propto \mathbb{E}_{\mathbf{s}\sim p_{\phi_t}^S}\left[\frac{1}{p_{\phi_t}^S(\mathbf{s})}\log q_\phi^G(\mathbf{s})\right]\end{aligned}$$

where we use the fact that the uniform distribution $U_{\mathcal{S}}(\mathbf{s})$ has constant density for all states in $\mathcal{S}$. However, computing this density $p_{\phi_t}^S(\mathbf{s})$ requires marginalizing out the MDP dynamics, which requires an accurate model of both the dynamics and the goal-conditioned policy.

We avoid needing to model the entire MDP process by approximating $p_{\phi_t}^S(\mathbf{s})$ with our previous learned generative model: $p_{\phi_t}^S(\mathbf{s}) \approx q_{\phi_t}^G(\mathbf{s})$. We therefore weight each state by the following weight function

$$w_{t,\alpha}(\mathbf{s}) \doteq q_{\phi_t}^G(\mathbf{s})^\alpha, \quad \alpha < 0. \tag{3.3}$$

where $\alpha$ is a hyperparameter that controls how heavily we weight each state. If our approximation $q_{\phi_t}^G$ is exact, we can choose $\alpha = -1$ and recover the exact IS procedure described above. If $\alpha = 0$, then this skew step has no effect. By choosing intermediate values of $\alpha$, we trade off the reliability of our estimate $q_{\phi_t}^G(\mathbf{s})$ with the speed at which we want to increase the goal distribution entropy.

**Variance Reduction** As described, this procedure relies on IS, which can have high variance, particularly if $q_{\phi_t}^G(\mathbf{s}) \approx 0$. We therefore choose a class of generative models where the probabilities are prevented from collapsing to zero, as we will describe in Section 3.2.3 where we provide generative model details. To further reduce the variance, we train $q_{\phi_{t+1}}^G$ with sampling importance resampling (SIR) [185] rather than IS. Rather than sampling from $p_{\phi_t}^S$ and weighting the update from each sample by $w_{t,\alpha}$, SIR explicitly defines a skewed empirical distribution as

$$\begin{aligned}p_{\text{skewed}_t}(\mathbf{s}) &\doteq \frac{1}{Z_\alpha} w_{t,\alpha}(\mathbf{s})\delta(\mathbf{s} \in \{\mathbf{s}_n\}_{n=1}^N) \\ Z_\alpha &= \sum_{n=1}^N w_{t,\alpha}(\mathbf{s}_n),\ \mathbf{s}_n \overset{\text{iid}}{\sim} p_{\phi_t}^S,\end{aligned} \tag{3.4}$$

where $\delta$ is the indicator function and $Z_\alpha$ is the normalizing coefficient. We note that computing $Z_\alpha$ adds little computational overhead, since all of the weights already need to be computed. We then fit the generative model at the next iteration $q_{\phi_{t+1}}^G$ to $p_{\text{skewed}_t}$ using standard MLE. We found that using SIR resulted in significantly lower variance than IS. See **??** C.2.2.2 for this comparision.

**Goal Sampling Alternative** Because $q^G_{\phi_{t+1}} \approx p_{\text{skewed}_t}$, at iteration $t+1$, one can sample goals from either $q^G_{\phi_{t+1}}$ or $p_{\text{skewed}_t}$. Sampling goals from $p_{\text{skewed}_t}$ may be preferred if sampling from the learned generative model $q^G_{\phi_{t+1}}$ is computationally or otherwise challenging. In either case, one still needs to train the generative model $q^G_{\phi_t}$ to create $p_{\text{skewed}_t}$. In our experiments, we found that both methods perform well.

**Summary** Overall, Skew-Fit collects states from the environment and resamples each state in proportion to Equation (3.3) so that low-density states are resampled more often. Skew-Fit is shown in Figure 3.10 and summarized in Algorithm 3. We now discuss conditions under which Skew-Fit converges to the uniform distribution.

**Algorithm 3** Skew-Fit

1: **for** Iteration $t = 1, 2, \ldots$ **do**
2: Collect $N$ states $\{\mathbf{s}_n\}_{n=1}^N$ by sampling goals from $q^G_{\phi_t}$ (or $p_{\text{skewed}_{t-1}}$) and running goal-conditioned policy.
3: Construct skewed distribution $p_{\text{skewed}_t}$ (Equation (3.3) and Equation (3.4)).
4: Fit $q^G_{\phi_{t+1}}$ to skewed distribution $p_{\text{skewed}_t}$ using MLE.
5: **end for**

#### 3.2.2.2 Skew-Fit Analysis

This section provides conditions under which $q^G_{\phi_t}$ converges in the limit to the uniform distribution over the state space $\mathcal{S}$. We consider the case where $N \to \infty$, which allows us to study the limit behavior of the goal distribution $p_{\text{skewed}_t}$. Our most general result is stated as follows:

**Lemma 1.** *Let $\mathcal{S}$ be a compact set. Define the set of distributions $\mathcal{Q} = \{p : \text{support}(p) \subseteq \mathcal{S}\}$. Let $\mathcal{F} : \mathcal{Q} \mapsto \mathcal{Q}$ be continuous with respect to the pseudometric $d_{\mathcal{H}}(p, q) \doteq |\mathcal{H}(p) - \mathcal{H}(q)|$ and $\mathcal{H}(\mathcal{F}(p)) \geq \mathcal{H}(p)$ with equality if and only if $p$ is the uniform probability distribution on $\mathcal{S}$, denoted as $U_{\mathcal{S}}$. Define the sequence of distributions $P = (p_1, p_2, \ldots)$ by starting with any $p_1 \in \mathcal{Q}$ and recursively defining $p_{t+1} = \mathcal{F}(p_t)$. The sequence $P$ converges to $U_{\mathcal{S}}$ with respect to $d_{\mathcal{H}}$. In other words, $\lim_{t\to 0} |\mathcal{H}(p_t) - \mathcal{H}(U_{\mathcal{S}})| \to 0$.*

*Proof.* See **??** C.2.1.1. □

We will apply Lemma 1 to be the map from $p_{\text{skewed}_t}$ to $p_{\text{skewed}_{t+1}}$ to show that $p_{\text{skewed}_t}$ converges to $U_{\mathcal{S}}$. If we assume that the goal-conditioned policy and generative model learning procedure are well behaved (i.e., the maps from $q^G_{\phi_t}$ to $p^S_{\phi_t}$ and from $p_{\text{skewed}_t}$ to $q^G_{\phi_{t+1}}$ are continuous), then to apply Lemma 1, we only need to show that $\mathcal{H}(p_{\text{skewed}_t}) \geq \mathcal{H}(p^S_{\phi_t})$ with equality if and only if $p^S_{\phi_t} = U_{\mathcal{S}}$. For the simple case when $q^G_{\phi_t} = p^S_{\phi_t}$ identically at each

iteration, we prove the convergence of Skew-Fit true for any value of $\alpha \in [-1, 0)$ in **??** C.2.1.3. However, in practice, $q^G_{\phi_t}$ only approximates $p^S_{\phi_t}$. To address this more realistic situation, we prove the following result:

**Lemma 2.** *Given two distribution $p^S_{\phi_t}$ and $q^G_{\phi_t}$ where $p^S_{\phi_t} \ll q^G_{\phi_t}$ [7] and*

$$\text{Cov}_{\mathbf{s} \sim p^S_{\phi_t}} \left[ \log p^S_{\phi_t}(\mathbf{s}), \log q^G_{\phi_t}(\mathbf{s}) \right] > 0, \tag{3.5}$$

*define the $p_{skewed_t}$ as in Equation (3.4) and take $N \to \infty$. Let $\mathcal{H}_\alpha(\alpha)$ be the entropy of $p_{skewed_t}$ for a fixed $\alpha$. Then there exists a constant $a < 0$ such that for all $\alpha \in [a, 0)$,*

$$\mathcal{H}(p_{skewed_t}) = \mathcal{H}_\alpha(\alpha) > \mathcal{H}(p^S_{\phi_t}).$$

*Proof.* See **??** C.2.1.2. □

This lemma tells us that our generative model $q^G_{\phi_t}$ does not need to exactly fit the sampled states. Rather, we merely need the log densities of $q^G_{\phi_t}$ and $p^S_{\phi_t}$ to be correlated, which we expect to happen frequently with an accurate goal-conditioned policy, since $p^S_{\phi_t}$ is the set of states seen when trying to reach goals from $q^G_{\phi_t}$. In this case, if we choose negative values of $\alpha$ that are small enough, then the entropy of $p_{\text{skewed}_t}$ will be higher than that of $p^S_{\phi_t}$. Empirically, we found that $\alpha$ values as low as $\alpha = -1$ performed well.

In summary, $p_{\text{skewed}_t}$ converges to $U_\mathcal{S}$ under certain assumptions. Since we train each generative model $q^G_{\phi_{t+1}}$ by fitting it to $p_{\text{skewed}_t}$ with MLE, $q^G_{\phi_t}$ will also converge to $U_\mathcal{S}$.

### 3.2.3 Training Goal-Conditioned Policies with Skew-Fit

Thus far, we have presented Skew-Fit assuming that we have access to a goal-reaching policy, allowing us to separately analyze how we can maximize $\mathcal{H}(\mathbf{G})$. However, in practice we do not have access to such a policy, and this section discusses how we concurrently train a goal-reaching policy.

Maximizing $I(\mathbf{s}; \mathbf{G})$ can be done by simultaneously performing Skew-Fit and training a goal-conditioned policy to minimize $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$, or, equivalently, maximize $-\mathcal{H}(\mathbf{G} \mid \mathbf{S})$. Maximizing $-\mathcal{H}(\mathbf{G} \mid \mathbf{S})$ requires computing the density $\log p(\mathbf{G} \mid \mathbf{s})$, which may be difficult to compute without strong modeling assumptions. However, for any distribution $q$, the following lower bound on $-\mathcal{H}(\mathbf{G} \mid \mathbf{S})$:

$$\begin{aligned} -\mathcal{H}(\mathbf{G} \mid \mathbf{S}) &= \mathbb{E}_{(\mathbf{G},\mathbf{s}) \sim q} \left[ \log q(\mathbf{G} \mid \mathbf{s}) \right] + D_{\text{KL}} \left( p \middle|\middle| q \right) \\ &\geq \mathbb{E}_{(\mathbf{G},\mathbf{s}) \sim q} \left[ \log q(\mathbf{G} \mid \mathbf{s}) \right], \end{aligned}$$

where $D_{\text{KL}}$ denotes Kullback–Leibler divergence as discussed by Barber and Agakov [9]. Thus, to minimize $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$, we train a policy to maximize the reward

$$r(\mathbf{s}, \mathbf{G}) = \log q(\mathbf{G} \mid \mathbf{s}).$$

[7] $p \ll q$ means that $p$ is absolutely continuous with respect to $q$, i.e. $p(\mathbf{s}) = 0 \implies q(\mathbf{s}) = 0$.

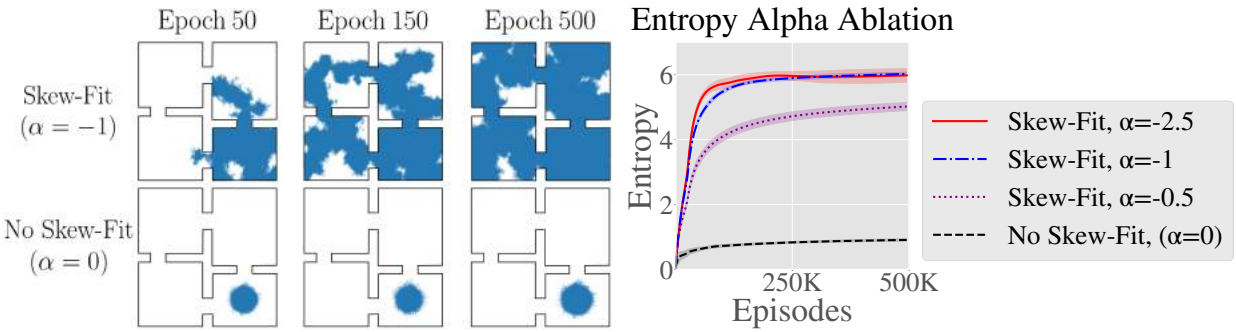



**Figure 3.11:** Illustrative example of Skew-Fit on a 2D navigation task. (Left) Visited state plot for Skew-Fit with $\alpha = -1$ and uniform sampling, which corresponds to $\alpha = 0$. (Right) The entropy of the goal distribution per iteration, mean and standard deviation for 9 seeds. Entropy is calculated via discretization onto an 11x11 grid. Skew-Fit steadily increases the state entropy, reaching full coverage over the state space.

The RL algorithm we use is reinforcement learning with imagined goals (RIG) [155], though in principle any goal-conditioned method could be used. RIG is an efficient off-policy goal-conditioned method that solves vision-based RL problems in a learned latent space. In particular, RIG fits a $\beta$-VAE [94] and uses it to encode observations and goals into a latent space, which it uses as the state representation. RIG also uses the $\beta$-VAE to compute rewards, $\log q(\mathbf{G} \mid \mathbf{s})$. Unlike RIG, we use the goal distribution from Skew-Fit to sample goals for exploration and for relabeling goals during training [4]. Since RIG already trains a generative model over states, we reuse this $\beta$-VAE for the generative model $q_\phi^G$ of Skew-Fit. To make the most use of the data, we train $q_\phi^G$ on all visited state rather than only the terminal states, which we found to work well in practice. To prevent the estimated state likelihoods from collapsing to zero, we model the posterior of the $\beta$-VAE as a multivariate Gaussian distribution with a fixed variance and only learn the mean. We summarize RIG and provide details for how we combine Skew-Fit and RIG in **??** C.2.3.4 and describe how we estimate the likelihoods given the $\beta$-VAE in **??** C.2.3.1.

### 3.2.4 Experiments

Our experiments study the following questions: **(1)** Does Skew-Fit empirically result in a goal distribution with increasing entropy? **(2)** Does Skew-Fit improve exploration for goal-conditioned RL? **(3)** How does Skew-Fit compare to prior work on choosing goals for vision-based, goal-conditioned RL? **(4)** Can Skew-Fit be applied to a real-world, vision-based robot task?

**Does Skew-Fit Maximize Entropy?** To see the effects of Skew-Fit on goal distribution entropy in isolation of learning a goal-reaching policy, we study an idealized example where the policy is a near-perfect goal-reaching policy. The environment consists of four rooms [201].

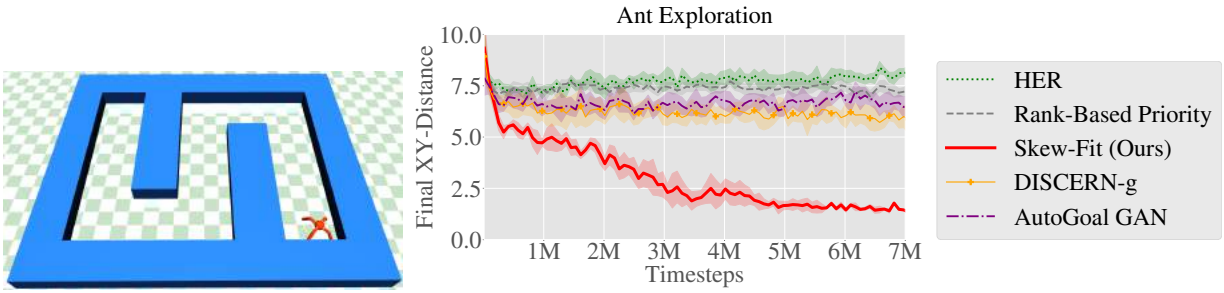



**Figure 3.12:** (Left) Ant navigation environment. (Right) Evaluation on reaching target XY position. We show the mean and standard deviation of 6 seeds. Skew-Fit significantly outperforms prior methods on this exploration task.

At the beginning of an episode, the agent begins in the bottom-right room and samples a goal from the goal distribution $q_{\phi_t}^G$. To simulate stochasticity of the policy and environment, we add a Gaussian noise with standard deviation of 0.06 units to this goal, where the entire environment is $11 \times 11$ units. The policy reaches the state that is closest to this noisy goal and inside the rooms, giving us a state sample $\mathbf{s}_n$ for training $q_{\phi_t}^G$. Due to the relatively small noise, the agent cannot rely on this stochasticity to explore the different rooms and must instead learn to set goals that are progressively farther and farther from the initial state. We compare multiple values of $\alpha$, where $\alpha = 0$ corresponds to not using Skew-Fit. The $\beta$-VAE hyperparameters used to train $q_{\phi_t}^G$ are given in **??** C.2.3.2. As seen in Figure 3.11, sampling uniformly from previous experience ($\alpha = 0$) to set goals results in a policy that primarily sets goal near the initial state distribution. In contrast, Skew-Fit results in quickly learning a high entropy, near-uniform distribution over the state space.

**Exploration with Skew-Fit** We next evaluate Skew-Fit while concurrently learning a goal-conditioned policy on a task with state inputs, which enables us study exploration performance independently of the challenges with image observations. We evaluate on a task that requires training a simulated quadruped "ant" robot to navigate to different XY positions in a labyrinth, as shown in Figure 3.12. The reward is the negative distance to the goal XY-position, and additional environment details are provided in Appendix C.2.4. This task presents a challenge for goal-directed exploration: the set of valid goals is unknown due to the walls, and random actions do not result in exploring locations far from the start. Thus, Skew-Fit must set goals that meaningfully explore the space while simultaneously learning to reach those goals.

We use this domain to compare Skew-Fit to a number of existing goal-sampling methods. We compare to the relabeling scheme described in the hindsight experience replay (labeled **HER**). We compare to curiosity-driven prioritization (**Ranked-Based Priority**) [232], a variant of HER that samples goals for relabeling based on their ranked likelihoods. Held et al. [93] samples goals from a GAN based on the difficulty of reaching the goal. We compare against this method by replacing $q_\phi^G$ with the GAN and label it **AutoGoal GAN**. We also

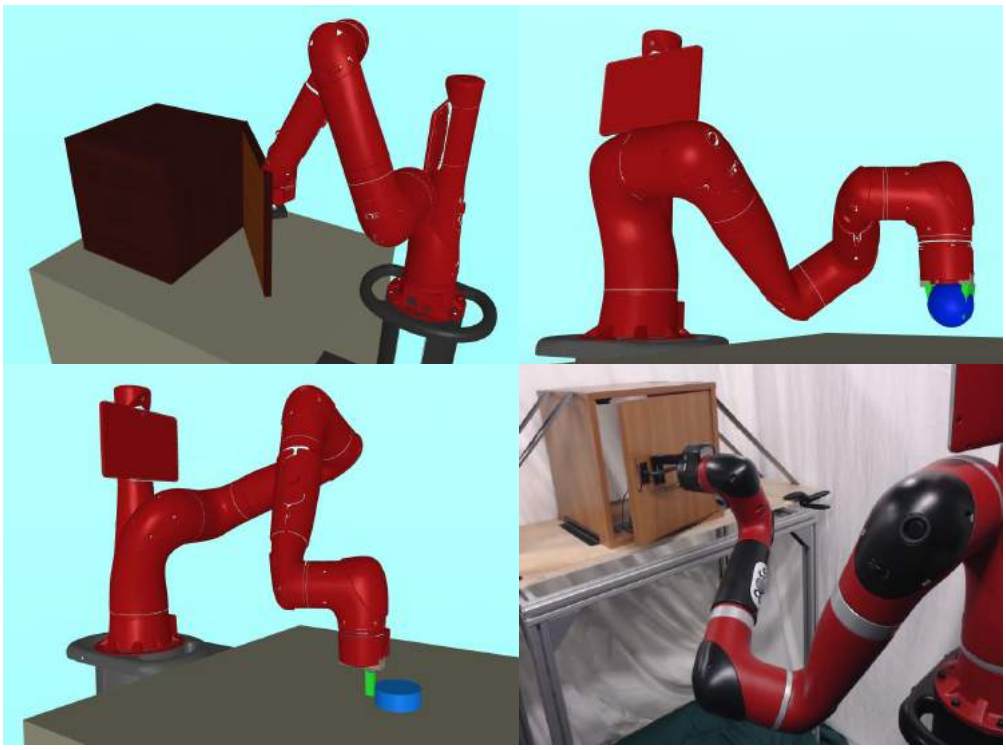


**Figure 3.13:** We evaluate on these continuous control tasks, from left to right: *Visual Door*, a door opening task; *Visual Pickup*, a picking task; *Visual Pusher*, a pushing task; and *Real World Visual Door*, a real world door opening task. All tasks are solved from images and without any task-specific reward. See Appendix C.2.4 for details.

compare to the non-parametric goal proposal mechanism proposed by [217], which we label **DISCERN-g**. Lastly, to demonstrate the difficulty of the exploration challenge in these domains, we compare to **#-Exploration** [203], an exploration method that assigns bonus rewards based on the novelty of new states. We train the goal-conditioned policy for each method using soft actor critic (SAC) [84]. Implementation details of SAC and the prior works are given in **??** C.2.3.3.

We see in Figure 3.12 that Skew-Fit is the only method that makes significant progress on this challenging labyrinth locomotion task. The prior methods on goal-sampling primarily set goals close to the start location, while the extrinsic exploration reward in #-Exploration dominated the goal-reaching reward. These results demonstrate that Skew-Fit accelerates exploration by setting diverse goals in tasks with unknown goal spaces.

**Vision-Based Continuous Control Tasks** We now evaluate Skew-Fit on a variety of image-based continuous control tasks, where the policy must control a robot arm using only image observations, there is no state-based or task-specific reward, and Skew-Fit must directly set image goals. We test our method on three different image-based simulated continuous

control tasks released by the authors of RIG [155]: *Visual Door*, *Visual Pusher*, and *Visual Pickup*. These environments contain a robot that can open a door, push a puck, and lift up a ball to different configurations, respectively. To our knowledge, these are the only goal-conditioned, vision-based continuous control environments that are publicly available and experimentally evaluated in prior work, making them a good point of comparison. See Figure 3.13 for visuals and Appendix C.2.3 for environment details. The policies are trained in a completely unsupervised manner, without access to any prior information about the image-space or any pre-defined goal-sampling distribution. To evaluate their performance, we sample goal images from a uniform distribution over valid states and report the agent's final distance to the corresponding simulator states (e.g., distance of the object to the target object location), but the agent never has access to this true uniform distribution nor the ground-truth state information during training. While this evaluation method is only practical in simulation, it provides us with a quantitative measure of a policy's ability to reach a broad coverage of goals in a vision-based setting.

We compare Skew-Fit to a number of existing methods on this domain. First, we compare to the methods described in the previous experiment (HER, Rank-Based Priority, #-Exploration, Autogoal GAN, and DISCERN-g). These methods that we compare to were developed in non-vision, state-based environments. To ensure a fair comparison across methods, we combine these prior methods with a policy trained using RIG. We additionally compare to Hazan et al. [91], an exploration method that assigns bonus rewards based on the likelihood of a state (labeled **Hazan et al.**). Next, we compare to **RIG** without Skew-Fit. Lastly, we compare to **DISCERN** [217], a vision-based method which uses a non-parametric clustering approach to sample goals and an image discriminator to compute rewards.

We see in Figure 3.14 that Skew-Fit significantly outperforms prior methods both in terms of task performance and sample complexity. The most common failure mode for prior methods is that the goal distributions collapse, resulting in the agent learning to reach only a fraction of the state space, as shown in Figure 3.9. For comparison, additional samples of $q_\phi^G$ when trained with and without Skew-Fit are shown in **??** C.2.2.3. Those images show that without Skew-Fit, $q_\phi^G$ produces a small, non-diverse distribution for each environment: the object is in the same place for pickup, the puck is often in the starting position for pushing, and the door is always closed. In contrast, Skew-Fit proposes goals where the object is in the air and on the ground, where the puck positions are varied, and the door angle changes.

We can see the effect of these goal choices by visualizing more example rollouts for RIG and Skew-Fit. These visuals, shown in Figure C.10 of **??** C.2.2.3, show that RIG only learns to reach states close to the initial position, while Skew-Fit learns to reach the entire state space. For a quantitative comparison, Figure 3.15 shows the cumulative total exploration pickups for each method. From the graph, we see that many methods have a near-constant rate of object lifts throughout all of training. Skew-Fit is the only method that significantly increases the rate at which the policy picks up the object during exploration, suggesting that only Skew-Fit sets goals that encourage the policy to interact with the object.

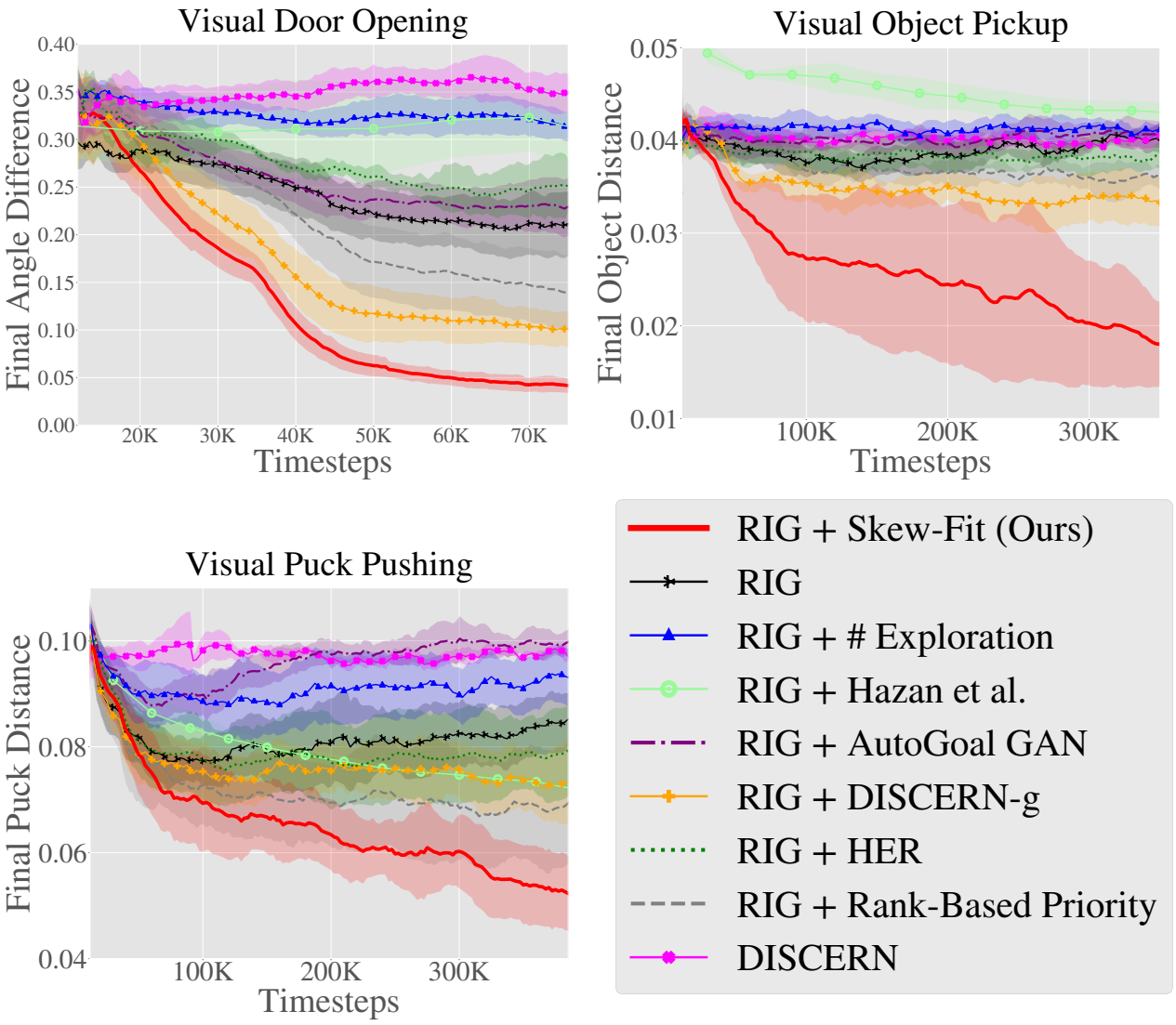



**Figure 3.14:** Learning curves for simulated continuous control tasks. Lower is better. We show the mean and standard deviation of 6 seeds and smooth temporally across 50 epochs within each seed. Skew-Fit consistently outperforms RIG and various prior methods. See text for description of each method.

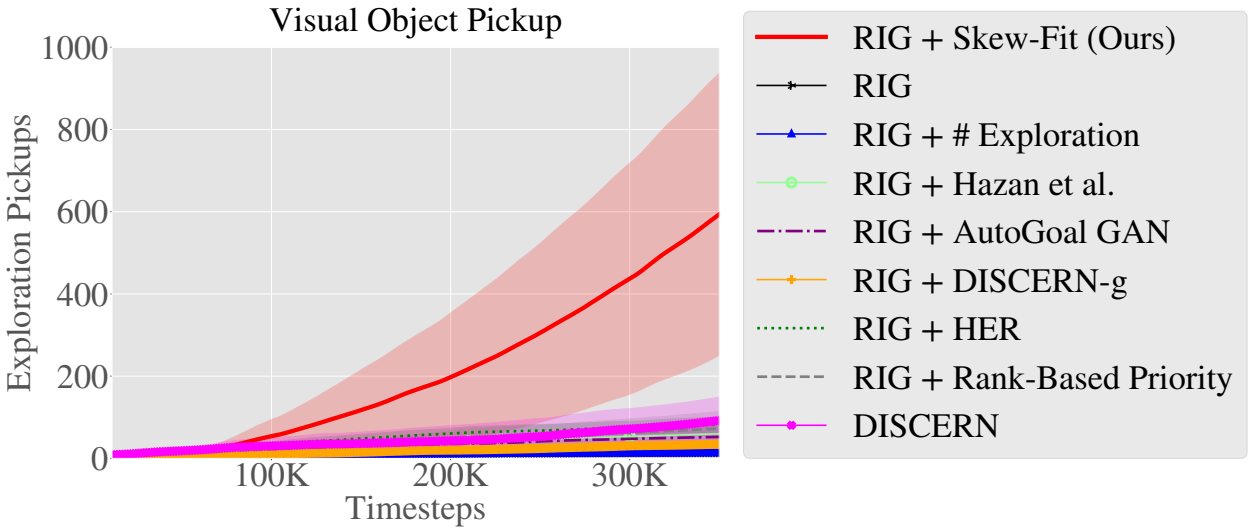



**Figure 3.15:** Cumulative total pickups during exploration for each method. Prior methods fail to pay attention to the object: the rate of pickups hardly increases past the first 100 thousand timesteps. In contrast, after seeing the object picked up a few times, Skew-Fit practices picking up the object more often by sampling the appropriate exploration goals.

**Real-World Vision-Based Robotic Manipulation** We also demonstrate that Skew-Fit scales well to the real world with a door opening task, *Real World Visual Door*, as shown in Figure 3.13. While a number of prior works have studied RL-based learning of door opening [106, 26], we demonstrate the first method for autonomous learning of door opening without a user-provided, task-specific reward function. As in simulation, we do not provide any goals to the agent and simply let it interact with the door, without any human guidance or reward signal. We train two agents using RIG and RIG with Skew-Fit. Every seven and a half minutes of interaction time, we evaluate on 5 goals and plot the cumulative successes for each method. Unlike in simulation, we cannot easily measure the difference between the policy's achieved and desired door angle. Instead, we visually denote a binary success/failure for each goal based on whether the last state in the trajectory achieves the target angle. As Figure 3.16 shows, standard RIG only starts to open the door after five hours of training. In contrast, Skew-Fit learns to occasionally open the door after three hours of training and achieves a near-perfect success rate after five and a half hours of interaction. Figure 3.16 also shows examples of successful trajectories from the Skew-Fit policy, where we see that the policy can reach a variety of user-specified goals. These results demonstrate that Skew-Fit is a promising technique for solving real world tasks without any human-provided reward function. Videos of Skew-Fit solving this task and the simulated tasks can be viewed on our website. [8]

[8] https://sites.google.com/view/skew-fit

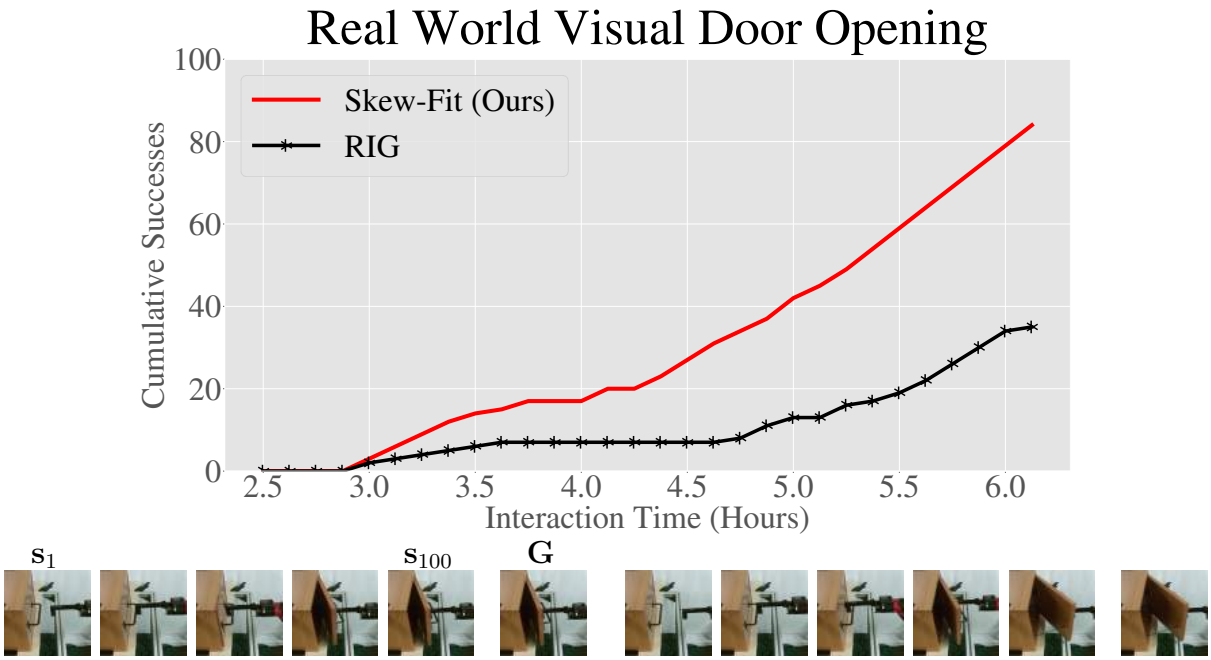



**Figure 3.16:** (Top) Learning curve for Real World Visual Door. Skew-Fit results in considerable sample efficiency gains over RIG on this real-world task. (Bottom) Each row shows the Skew-Fit policy starting from state $\mathbf{s}_1$ and reaching state $\mathbf{s}_{100}$ while pursuing goal $\mathbf{G}$. Despite being trained from only images without any user-provided goals during training, the Skew-Fit policy achieves the goal image provided at test-time, successfully opening the door.

**Additional Experiments** To study the sensitivity of Skew-Fit to the hyperparameter $\alpha$, we sweep $\alpha$ across the values $[-1, -0.75, -0.5, -0.25, 0]$ on the simulated image-based tasks. The results are in Appendix C.2.2 and demonstrate that Skew-Fit works across a large range of values for $\alpha$, and $\alpha = -1$ consistently outperform $\alpha = 0$ (i.e. outperforms no Skew-Fit). Additionally, Appendix C.2.3 provides a complete description our method hyperparameters, including network architecture and RL algorithm hyperparameters.

## 3.3 Conclusion

In this chapter, we presented new RL algorithms that can efficiently solve goal-conditioned, vision-based tasks without access to any ground truth state or reward functions. These methods train a generative model that is used for multiple purposes: we embed the state and goals using the encoder; we sample from the prior to generate goals for exploration; we also sample latents to retroactively relabel goals and rewards; and we use distances in the latent space for rewards to train a goal-conditioned value function. We show that these components culminate in a sample efficient algorithm that works directly from vision. As a result, we are able to apply our method to a variety of simulated visual tasks, including a

variable-object task that cannot be easily represented with a fixed length vector, as well as real world robotic tasks. We also presented a formal objective for self-supervised goal-directed exploration, allowing researchers to quantify and compare progress when designing algorithms that enable agents to autonomously learn. To optimize this objective, we presented Skew-Fit, an algorithm for training a generative model to approximate a uniform distribution over an initially unknown set of valid states, using data obtained via goal-conditioned reinforcement learning, and our theoretical analysis gives conditions under which Skew-Fit converges to the uniform distribution. When such a model is used to choose goals for exploration and to relabeling goals for training, the resulting method results in much better coverage of the state space, enabling our method to explore effectively. Our experiments show that when we concurrently train a goal-reaching policy using self-generated goals, Skew-Fit produces quantifiable improvements on simulated robotic manipulation tasks, and can be used to learn a door opening skill to reach a 95% success rate directly on a real-world robot, without any human-provided reward supervision.

This chapter concludes our discussion on generating goals and autonomously practicing to reach them. Since the original publication of RIG [155] and Skew-Fit [173], a number of papers have extended and complimented the framework presented here in exciting ways. For example, Pitis et al. [170] empirically demonstrates that exploration can be accelerated by much more aggressively skewing the state, i.e. by conceptually setting $\alpha$ to values even less than $-1$. Colas et al. [37] demonstrates that using language-based goals enables agents to generate more out-of-distribution goals by leveraging language's compositionality. Lee et al. [130] discusses the use of prior information to match a target distribution rather than a uniform state distribution. Incorporating modularity into goal setting, analyzing and accelerating the rate of convergence of these goal-directed exploration methods, and incorporating prior information are all promising directions for future research.

Goal-conditioned reinforcement learning provides a powerful mechanism for autonomously acquiring goal-reaching skills. These skills can be reused by simply providing a new goal to a policy, and in the next chapter, we discuss how we can enable policies to achieve longer-horizon tasks and optimize new rewards by combining goal-conditioned policies with planning-based methods.

# Chapter 4

# Reusing Goal-Directed Behavior

Thus far, we have presented methods for acquiring goal-conditioned skills by learning through direct interaction with the environment. However, solving complex and temporally extended sequential decision making requires more than just well-honed reactions. Agents that generalize effectively to new situations and new tasks must reuse their capabilities and reason about the consequences of their actions and solve new problems via planning. Accomplishing this entirely with model-free RL often proves challenging, as purely model-free learning does not inherently provide for temporal compositionality of skills.

Planning and trajectory optimization algorithms encode this temporal compositionality by design, but require accurately models with which to plan. When these models are specified manually, planning can be very powerful, but learning such models presents major obstacles: in complex environments with high-dimensional observations such as images, direct prediction of future observations presents a very difficult modeling problem [17, 162, 145, 31, 107, 7, 129], and model errors accumulate over time [154], making their predictions inaccurate in precisely those long-horizon settings where we most need the compositionality of planning methods. Can we obtain the benefits of temporal compositionality inherent in model-based planning, without the need to model the environment at the lowest level, in terms of both time and state representation?

One way to avoid modeling the environment in detail is to plan over *abstractions*: simplified representations of states and transitions on which it is easier to construct predictions and plans. *Temporal* abstractions allow planning at a coarser time scale, skipping over the high-frequency details and instead planning over higher-level subgoals, while *state* abstractions allow planning over a simpler representation of the state. Both make modeling and planning easier.

In this chapter, we study how model-free RL, and specifically goal-conditioned model-free RL, can be used to provide such abstraction for a model-based planner. At first glance, this might seem like a strange proposition, since model-free RL methods learn value functions and policies, not models. However, this is precisely what makes them ideal for abstracting away the complexity in temporally extended tasks with high-dimensional observations: by avoiding low-level (e.g., pixel-level) prediction, model-free RL can acquire behaviors that manipulate

these low-level observations without needing to predict them explicitly. This leaves the planner free to operate at a higher level of abstraction, reasoning about the capabilities of low-level model-free policies.

We begin by presenting temporal difference models, a method for reusing goal-conditioned value functions in a planning-based framework for domains with relatively simple state spaces. We then discuss how to extend this work to more complex observation spaces such as images.

## 4.1 Temporal Difference Models

To arrive at a method that combines the strengths of model-free and model-based RL, we study a variant of goal-conditioned value functions [202, 190, 4]. As discussed in Section 2.1, goal-conditioned value functions learn to predict the value function for every possible goal state. That is, they answer the following question: what is the expected reward for reaching a particular state, given that the agent is attempting (as optimally as possible) to reach it? The particular choice of reward function determines what such a method actually does, but rewards based on distances to a goal hint at a connection to model-based learning: if we can predict how easy it is to reach any state from any current state, we must have some kind of understanding of the underlying "physics." In fact, for a specific choice of reward and horizon, the value function corresponds directly to a model, while for larger horizons, it more closely resembles model-free approaches. Extension toward more model-free learning is thus achieved by acquiring "multi-step models" that can be used to plan over progressively coarser temporal resolutions, eventually arriving at a fully model-free formulation.

We presents a new RL algorithm that makes use of this connection between model-based and model-free learning to learn a specific type of goal-conditioned value function, which we call a temporal difference model (TDM). This value function can be learned very efficiently, with sample complexities that are competitive with model-based RL, and can then be used with an MPC-like method to accomplish desired tasks. Our empirical experiments demonstrate that this method achieves substantially better sample complexity than fully model-free learning on a range of challenging continuous control tasks, while outperforming purely model-based methods in terms of final performance. Furthermore, the connection that our method elucidates between model-based and model-free learning may lead to a range of interesting future methods. We begin by briefly summarizing common approaches to model-based RL.

### 4.1.1 Model-based RL and optimal control.

In model-based RL, the aim is to train a model of the form $f(\mathbf{s}_t, \mathbf{a}_t)$ to predict the next state $\mathbf{s}_{t+1}$. Once trained, this model can be used to choose actions, either by backpropagating reward gradients into a policy, or planning directly through the model. In the latter case, a particularly effective method for employing a learned model is model-predictive control (MPC), where a new action plan is generated at each time step, and the first action of that

plan is executed, before replanning begins from scratch. MPC can be formalized as the following optimization problem:

$$\mathbf{a}_t = \arg\max_{a_{t:t+T}} \sum_{i=t}^{t+T} r(s_i, a_i) \text{ where } \mathbf{s}_{i+1} = f(s_i, a_i) \ \forall \ i \in \{t, ..., t+T-1\}. \tag{4.1}$$

We can also write the dynamics constraint in the above equation in terms of an implicit dynamics, according to

$$\mathbf{a}_t = \arg\max_{a_{t:t+T}, s_{t+1:t+T}} \sum_{i=t}^{t+T} r(s_i, a_i) \text{ such that } C(s_i, a_i, \mathbf{s}_{i+1}) = 0 \ \forall \ i \in \{t, ..., t+T-1\}, \tag{4.2}$$

where $C(s_i, a_i, \mathbf{s}_{i+1}) = 0$ if and only if $\mathbf{s}_{i+1} = f(s_i, a_i)$. This implicit version will be important in understanding the connection between model-based and model-free RL.

### 4.1.2 Temporal Difference Model Learning

In this section, we introduce a type of goal-conditioned value functions called temporal difference models (TDMs) that provide a direct connection to model-based RL. We will first motivate this connection by relating the model-based MPC optimizations in Equations (4.1) and (4.2) to goal-conditioned value functions, and then present our temporal difference model derivation, which extends this connection from a purely model-based setting into one that becomes increasingly model-free.

#### 4.1.2.1 From Goal-Conditioned Value Functions to Models

Let us consider the choice of reward function for the goal conditioned value function. Although a variety of options have been explored in the literature [202, 190, 4], a particularly intriguing connection to model-based RL emerges if we set $\mathcal{G} = \mathcal{S}$, such that $g \in \mathcal{G}$ corresponds to a *goal state* $\mathbf{g} \in \mathcal{S}$, and we consider distance-based reward functions $r_d$ of the following form:

$$r_d(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{g}) = -D(\mathbf{s}_{t+1}, \mathbf{g}),$$

where $D(\mathbf{s}_{t+1}, \mathbf{g})$ is a distance, such as the Euclidean distance $D(\mathbf{s}_{t+1}, \mathbf{g}) = \|\mathbf{s}_{t+1} - \mathbf{g}\|_2$. If $\gamma = 0$, we have $Q(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}) = -D(\mathbf{s}_{t+1}, \mathbf{g})$ at convergence of $Q$-learning, which means that $Q(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}) = 0$ implies that $\mathbf{s}_{t+1} = \mathbf{g}$. Plug this $Q$-function into the model-based planning optimization in Equation (4.2), denoting the task control reward as $r_c$, such that the solution to

$$\mathbf{a}_t = \arg\max_{a_{t:t+T}, s_{t+1:t+T}} \sum_{i=t}^{t+T} r_c(s_i, a_i) \text{ such that } Q(s_i, a_i, \mathbf{s}_{i+1}) = 0 \ \forall \ i \in \{t, ..., t+T-1\} \tag{4.3}$$

yields a model-based plan. We have now derived a precise connection between model-free and model-based RL, in that model-free learning of goal-conditioned value functions can be used

to directly produce an implicit model that can be used with MPC-based planning. However, this connection by itself is not very useful: the resulting implicit model is fully model-based, and does not provide any kind of long-horizon capability. In the next section, we show how to extend this connection into the long-horizon setting by introducing the temporal difference model (TDM).

#### 4.1.2.2 Long-Horizon Learning with Temporal Difference Models

If we consider the case where $\gamma > 0$, the optimization in Equation (4.3) no longer corresponds to any optimal control method. In fact, when $\gamma = 0$, Q-values have well-defined units: units of distance between states. For $\gamma > 0$, no such interpretation is possible. The key insight in temporal difference models is to introduce a different mechanism for aggregating long-horizon rewards. Instead of evaluating Q-values as discounted sums of rewards, we introduce an additional input $t$, which represents the planning horizon, and define the $Q$-learning recursion as

$$Q(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}, t) = \mathbb{E}_{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}[-D(\mathbf{s}_{t+1}, \mathbf{g})\mathbb{1}[t=0] + \max_a Q(\mathbf{s}_{t+1}, a, \mathbf{g}, t-1)\mathbb{1}[t\neq 0]]. \tag{4.4}$$

The $Q$-function uses a reward of $-D(\mathbf{s}_{t+1}, \mathbf{g})$ when $t = 0$ (at which point the episode terminates), and decrements $t$ by one at every other step. Since this is still a well-defined $Q$-learning recursion, it can be optimized with off-policy data and, just as with goal-conditioned value functions, we can resample new goals $\mathbf{g}$ and new horizons $t$ for each tuple $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$, even ones that were not actually used when the data was collected. In this way, the TDM can be trained very efficiently, since every tuple provides supervision for every possible goal and every possible horizon.

The intuitive interpretation of the TDM is that it tells us how close the agent will get to a given goal state $\mathbf{g}$ after $t$ time steps, *when it is attempting to reach that state in $t$ steps*. Alternatively, TDMs can be interpreted as Q-values in a finite-horizon MDP, where the horizon is determined by $t$. For the case where $t = 0$, TDMs effectively learn a model, allowing TDMs to be incorporated into a variety of planning and optimal control schemes at test time as in Equation (4.3). Thus, we can view TDM learning as an interpolation between model-based and model-free learning, where $t = 0$ corresponds to the single-step prediction made in model-based learning and $t > 0$ corresponds to the long-term prediction made by typical $Q$-functions. While the correspondence to models is not the same for $t > 0$, if we only care about the reward at every $K$ step, then we can recover a correspondence by replace Equation (4.3) with

$$\begin{aligned} \mathbf{a}_t = & \arg\max_{a_{t:K:t+T}, s_{t+K:K:t+T}} \sum_{i=t,t+K,\ldots,t+T} r_c(s_i, a_i) \\ & \text{such that } Q(s_i, a_i, s_{i+K}, K-1) = 0 \;\forall\; i \in \{t, t+K, ..., t+T-K\}, \end{aligned} \tag{4.5}$$

where we only optimize over every $K^{\text{th}}$ state and action. As the TDM becomes effective for longer horizons, we can increase $K$ until $K = T$, and plan over only a single effective time

step:

$$\mathbf{a}_t = \underset{\mathbf{a}_t, a_{t+T}, \mathbf{s}_{t+T}}{\arg\max}\ r_c(\mathbf{s}_{t+T}, a_{t+T}) \text{ such that } Q(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+T}, T-1) = 0. \tag{4.6}$$

This formulation does result in some loss of generality, since we no longer optimize the reward at the intermediate steps. This limits the multi-step formulation to terminal reward problems, but does allow us to accommodate arbitrary reward functions on the terminal state $\mathbf{s}_{t+T}$, which still describes a broad range of practically relevant tasks. In the next section, we describe how TDMs can be implemented and used in practice for continuous state and action spaces.

### 4.1.3 Training and Using Temporal Difference Models

The TDM can be trained with any off-policy $Q$-learning algorithm, such as DQN [149], DDPG [138], NAF [77], and SDQN [146]. During off-policy $Q$-learning, TDMs can benefit from arbitrary relabeling of the goal states $g$ and the horizon $t$, given the same $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ tuples from the behavioral policy as done in [4]. This relabeling enables simultaneous, data-efficient learning of short-horizon and long-horizon behaviors for arbitrary goal states, unlike previously proposed goal-conditioned value functions that only learn for a single time scale, typically determined by a discount factor [190, 4]. In this section, we describe the design decisions needed to make practical a TDM algorithm.

#### 4.1.3.1 Reward Function Specification

$Q$-learning typically optimizes scalar rewards, but TDMs enable us to increase the amount of supervision available to the $Q$-function by using a vector-valued reward. Specifically, if the distance $D(s, \mathbf{g})$ factors additively over the dimensions, we can train a vector-valued $Q$-function that predicts per-dimension distance, with the reward function for dimension $j$ given by $-D_j(\mathbf{s}_j, \mathbf{g}_j)$. We use the $\ell_1$ norm in our implementation, which corresponds to absolute value reward $-|\mathbf{s}_j - \mathbf{g}_j|$. The resulting vector-valued $Q$-function can learn distances along each dimension separately, providing it with more supervision from each training point. Empirically, we found that this modifications provides a substantial boost in sample efficiency.

We can optionally make an improvement to TDMs if we know that the task reward $r_c$ depends only on some subset of the state or, more generally, state features. In that case, we can train the TDM to predict distances along only those dimensions or features that are used by $r_c$, which in practice can substantially simplify the corresponding prediction problem. In our experiments, we illustrate this property by training TDMs for pushing tasks that predict distances from an end-effector and pushed object, without accounting for internal joints of the arm, and similarly for various locomotion tasks.

#### 4.1.3.2 Policy Extraction with TDMs

While the TDM optimal control formulation Equation (4.6) drastically reduces the number of states and actions to be optimized for long-term planning, it requires solving a constrained

optimization problem, which is more computationally expensive than unconstrained problems. We can remove the need for a constrained optimization through a specific architectural decision in the design of the function approximator for $Q(\mathbf{s}, \mathbf{a}, \mathbf{g}, t)$. We define the $Q$-function as $Q(\mathbf{s}, \mathbf{a}, \mathbf{g}, t) = -\|f(\mathbf{s}, \mathbf{a}, \mathbf{g}, t) - \mathbf{g}\|$, where $f(\mathbf{s}, \mathbf{a}, \mathbf{g}, t)$ outputs a state vector. By training the TDM with a standard $Q$-learning method, $f(\mathbf{s}, \mathbf{a}, \mathbf{g}, t)$ is trained to explicitly predict the state that will be reached by a policy attempting to reach $\mathbf{g}$ in $t$ steps. This model can then be used to choose the action with fully explicit MPC as below, which also allows straightforward derivation of a multi-step version as in Equation (4.5).

$$\mathbf{a}_t = \arg\max_{\mathbf{a}_t, \mathbf{a}_{t+T}, \mathbf{s}_{t+T}} r_c(f(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+T}, T-1), \mathbf{a}_{t+T}) \tag{4.7}$$

In the case where the task is to reach a goal state $\mathbf{g}$, a simpler approach to extract a policy is to use the TDM directly:

$$\mathbf{a}_t = \arg\max_a Q(\mathbf{s}_t, \mathbf{a}, \mathbf{g}, T) \tag{4.8}$$

In our experiments, we use Equations (4.7) and (4.8) to extract a policy.

#### 4.1.3.3 Algorithm Summary

**Algorithm 4** Temporal Difference Model Learning

**Require:** Task reward function $r_c(\mathbf{s}, \mathbf{a})$, parameterized TDM $Q_w(\mathbf{s}, \mathbf{a}, \mathbf{g}, t)$, replay buffer $\mathcal{B}$
**for** $n = 0, ..., N-1$ episodes **do**
  $\mathbf{s}_0 \sim p(\mathbf{s}_0)$
  **for** $t = 0, ..., T-1$ time steps **do**
    $\mathbf{a}_t^* = \text{MPC}(r_c, \mathbf{s}_t, Q_w, T-t)$ {Eq. 4.5, Eq. 4.6, Eq. 4.7, or Eq. 4.8}
    $\mathbf{a}_t = \text{AddNoise}(\mathbf{a}_t^*)$ {Noisy exploration}
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_t, \mathbf{a}_t)$, and store $\{\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}\}$ in the replay buffer $\mathcal{B}$ {Step environment}
    **for** $i = 0, I-1$ iterations **do**
      Sample $M$ transitions $\{\mathbf{s}_m, \mathbf{a}_m, \mathbf{s}'_m\}$ from the replay $\mathcal{B}$.
      Relabel time horizons and goal states $t, \mathbf{g}_m$ {?? D.1.1.1}
      $y_m = -\|\mathbf{s}'_m - \mathbf{g}_m\| \mathbb{1}[t = 0] + \max_{\mathbf{a}} Q'(\mathbf{s}'_m, \mathbf{a}, \mathbf{g}_m, t-1) \mathbb{1}[t \neq 0]$
      $L(w) = \sum_m (Q_w(\mathbf{s}_m, \mathbf{a}_m, \mathbf{g}_m, t) - y_m)^2 / M$ {Compute the loss}
      Minimize$(w, L(w))$ {Optimize}
    **end for**
  **end for**
**end for**

The algorithm is summarized as Algorithm 4. A crucial difference from prior goal-conditioned value function methods [190, 4] is that our algorithm can be used to act according to an arbitrary terminal reward function $r_c$, both during exploration and at test time. Like other off-policy algorithms [149, 138], it consists of exploration and $Q$-function fitting. Noise is injected for exploration, and $Q$-function fitting uses standard $Q$-learning techniques, with target networks $Q'$ and experience replay [149, 138]. If we view the $Q$-function fitting as model

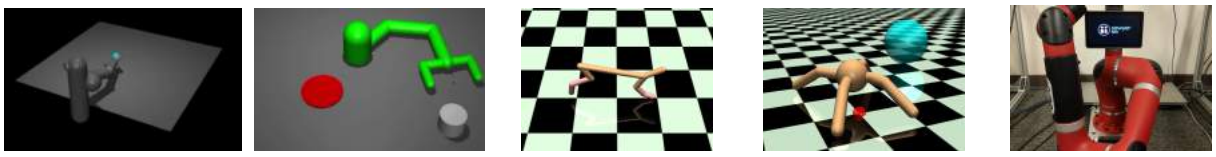
fitting, the algorithm also resembles iterative model-based RL, which alternates between collecting data using the learned dynamics model for planning [43] and fitting the model. Since we focus on continuous tasks, we use DDPG [138], though any $Q$-learning method could be used.

The computation cost of the algorithm is mostly determined by the number of updates to fit the $Q$-function per transition, $I$. In general, TDMs can benefit from substantially larger $I$ than classic model-free methods such as DDPG due to relabeling increasing the amount of supervision signals. In real-world applications such as robotics where we care most of the sample efficiency [78], the learning is often bottlenecked by the data collection rather than the computation, and therefore large $I$ values are usually not a significant problem and can continuously benefit from the acceleration in computation.

### 4.1.4 Experiments

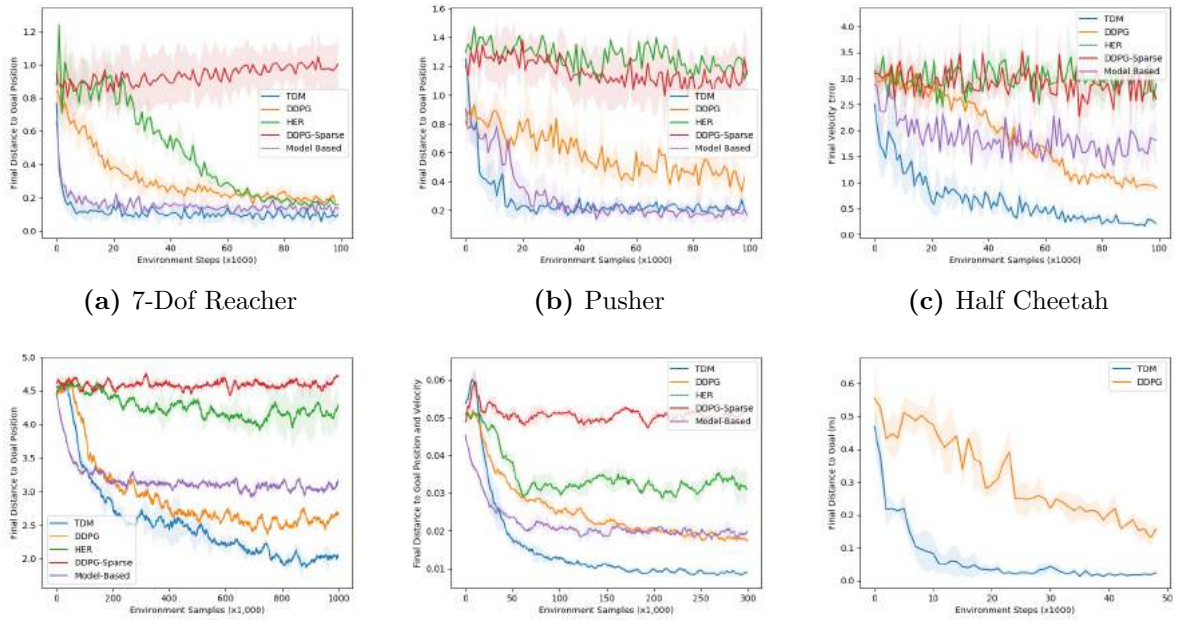
Our experiments examine how the sample efficiency and performance of TDMs compare to both model-based and model-free RL algorithms. We expect to have the efficiency of model-based RL but with less model bias. We also aim to study the importance of several key design decisions in TDMs, and evaluate the algorithm on a real-world robotic platform. For the model-free comparison, we compare to DDPG [138], which typically achieves the best sample efficiency on benchmark tasks [51]; HER, which uses goal-conditioned value functions [4]; and DDPG with the same sparse rewards of HER. For the model-based comparison, we compare to the model-based component in [153], a recent work that reports highly efficient learning with neural network dynamics models. Details of the baseline implementations are in the Appendix. We perform the comparison on five simulated tasks: (1) a 7 DoF arm reaching various random end-effector targets, (2) an arm pushing a puck to a target location, (3) a planar cheetah attempting to reach a goal velocity (either forward or backward), (4) a quadrupedal ant attempting to reach a goal position, and (5) an ant attempting to reach a goal position and velocity. The tasks are shown in Figure 4.1 and terminate when either the goal is reached or the time horizon is reached. The pushing task requires long-horizon reasoning to reach and push the puck. The cheetah and ant tasks require handling many contact discontinuities which is challenging for model-based methods, with the ant environment having particularly difficult dynamics given the larger state and action space. The ant position and velocity task presents a scenario where reward shaping as in traditional RL methods may not lead to optimal behavior, since one cannot maintain both a desired position and velocity. However, such a task can be very valuable in realistic settings. For example, if we want the ant to jump, we might instruct it to achieve a particular velocity at a particular location. We also tested TDMs on a real-world robot arm reaching end-effector positions, to study its applicability to real-world tasks.

For the simulated and real-world 7-DoF arm, our TDM is trained on all state components. For the pushing task, our TDM is trained on the hand and puck XY-position. For the half cheetah task, our TDM is trained on the velocity of the cheetah. For the ant tasks, our TDM



**(a)** 7-DoF Reacher **(b)** Pusher **(c)** Half Cheetah **(d)** Ant **(e)** Sawyer Robot

**Figure 4.1:** The tasks in our experiments: (a) reaching target locations, (b) pushing a puck to a random target, (c) training the cheetah to run at target velocities, (d) training an ant to run to a target position or a target position and velocity, and (e) reaching target locations (real-world Sawyer robot).

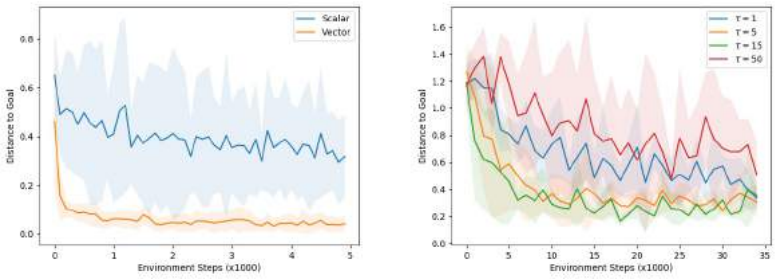



**(a)** 7-Dof Reacher **(b)** Pusher **(c)** Half Cheetah

**(d)** Ant: Position **(e)** Ant: Position and Velocity **(f)** Sawyer Robot (Real-world)

**Figure 4.2:** The comparison of TDM with model-free (DDPG, both with sparse and dense rewards), model-based, and goal-conditioned value functions (HER with sparse rewards) methods on various tasks. All plots show the final distance to the goal versus 1000 environment steps (not rollouts). The bold line shows the mean across 3 random seeds, and the shaded region show one standard deviation. Our method, which uses model-free learning, is generally more sample-efficient than model-free alternatives including DDPG and HER and improves upon the best model-based performance.

is trained on either the position or the position and velocity for the respective task. Full details are in the Appendix.




**(a)** Scalar vs Vectorized TDMs **(b)** TDMs with different $t_{max}$

**Figure 4.3:** Ablation experiments for (a) scalar vs. vectorized TDMs on 7-DoF simulated reacher task and (b) different $t_{max}$ on pusher task. The vectorized variant performs substantially better, while the horizon effectively interpolates between model-based and model-free learning.

#### 4.1.4.1 TDMs vs Model-Free, Mode-Based, and Direct Goal-Conditioned RL

The results are shown in Figure 4.2. When compared to the model-free baselines, the pure model-based method learns learns much faster on all the tasks. However, on the harder cheetah and ant tasks, its final performance is worse due to model bias. TDMs learn as quickly or faster than the model-based method, but also always learn policies that are as good as if not better than the model-free policies. Furthermore, TDMs requires fewer samples than the model-free baselines on ant tasks and drastically fewer samples on the other tasks. We also see that using HER does not lead to an improvement over DDPG. While we were initially surprised, we realized that a selling point of HER is that it can solve sparse tasks that would otherwise be unsolvable. In this paper, we were interested in improving the sample efficiency and not the feasibility of model-free reinforcement learning algorithms, and so we focused on tasks that DDPG could already solve. In these sorts of tasks, the advantage of HER over DDPG with a dense reward is not expected. To evaluate HER as a method to solve sparse tasks, we included the DDPG-Sparse baseline and we see that HER significantly outperforms it as expected. In summary, TDMs converge as fast or faster than model-based learning (which learns faster than the model-free baselines), while achieving final performance that is as good or better that the model-free methods on all tasks.

Lastly, we ran the algorithm on a 7-DoF Sawyer robotic arm to learn a real-world analogue of the reaching task. Figure 4.2f shows that the algorithm outperforms and learns with fewer samples than DDPG, our model-free baseline. These results show that TDMs can scale to real-world tasks.

#### 4.1.4.2 Ablation Studies

We discuss two key design choices for TDMs that provide substantially improved performance. First, Figure 4.3a examines the tradeoffs between the vectorized and scalar rewards. The results show that the vectorized formulation learns substantially faster than the naïve scalar

variant. Second, Figure 4.3b compares the learning speed for different horizon values $t_{max}$. Performance degrades when the horizon is too low, and learning becomes slower when the horizon is too high. These results suggest that TDMs perform well because they provide an intermediate temporal abstraction between one-step planning methods and infinite-horizon model-free methods.

These experiments were all conducted in domain with relatively low dimensional state spaces. In the next chapter, we discuss how one can apply TDMs to complex state representations, such as images, where simple distance metrics may no longer be effective.

## 4.2 Planning with Images

In this section, we study how we can extend the ideas from Section 4.1 to domains with complex observation spaces. A major challenge with this setup is the need to actually optimize over subgoals. In domains with high-dimensional observations such as images, this may require explicitly optimizing over image pixels. This optimization is challenging, as realistic images – and, in general, feasible states – typically form a thin, low-dimensional manifold within the larger space of possible state observation values [142]. To address this, we build abstractions of the state observation by learning a compact latent variable state representation, which makes it feasible to optimize over the goals in domains with high-dimensional observations, such as images, without explicitly optimizing over image pixels. The learned representation allows the planner to determine which subgoals actually represent feasible states, while the learned goal-conditioned value function tells the planner whether these states are reachable.

We evaluate our method on temporally extended tasks that require multistage reasoning and handling image observations. We discuss how the low-level goal-reaching policies themselves cannot solve these tasks effectively, as they do not plan over subgoals and therefore do not benefit from temporal compositionality. Planning without state representation learning also fails to perform these tasks, as optimizing directly over images results in invalid subgoals. By contrast, our method, which we call Latent Embeddings for Abstracted Planning (LEAP), is able to successfully determine suitable subgoals by searching in the latent representation space, and then reach these subgoals via the model-free policy.

### 4.2.1 Planning with Goal-Conditioned Policies

We aim to learn a model that can solve arbitrary long-horizon goal reaching tasks with high-dimensional observation and goal spaces, such as images. A model-free goal-conditioned reinforcement learning algorithm could, in principle, solve such a problem. However, as we will show in our experiments, in practice such methods produce overly greedy policies, which can accomplish short-term goals, but struggle with goals that are more temporally extended. We instead combine goal-conditioned policies trained to achieve subgoals with a planner that decomposes long-horizon goal-reaching tasks into $K$ shorter horizon subgoals. Specifically, our planner chooses the $K$ subgoals, $\mathbf{g}_1, \ldots, \mathbf{g}_K$, and a goal-reaching policy then attempts to

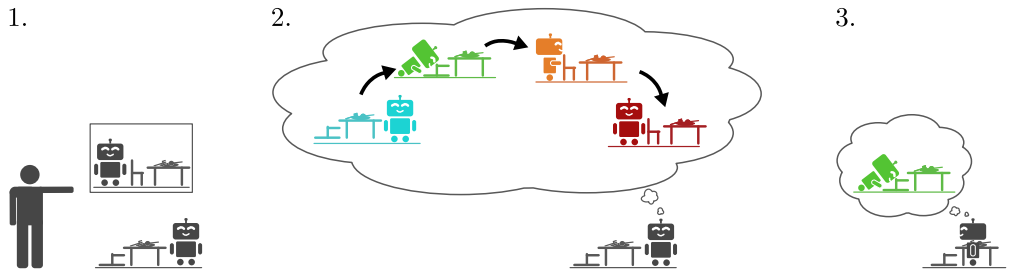



**Figure 4.4:** Summary of Latent Embeddings for Abstracted Planning (LEAP). (1) The planner is given a goal state. (2) The planner plans intermediate subgoals in a low-dimensional latent space. By planning in this latent space, the subgoals correspond to valid state observations. (3) The goal-conditioned policy then tries to reach the first subgoal. After $t_1$ time steps, the policy replans and repeats steps 2 and 3.

reach the first subgoal $\mathbf{g}_1$ in the first $t_1$ time steps, before moving onto the second goal $\mathbf{g}_2$, and so forth, as shown in Figure 4.4. This procedure only requires training a goal-conditioned policy to solve short-horizon tasks. Moreover, by planning appropriate subgoals, the agent can compose previously learned goal-reaching behavior to solve new, temporally extended tasks. The success of this approach will depend heavily on the choice of subgoals. In the sections below, we outline how one can measure the quality of the subgoals. Then, we address issues that arise when optimizing over these subgoals in high-dimensional state spaces such as images. Lastly, we summarize the overall method and provide details on our implementation.

#### 4.2.1.1 Planning over Subgoals

Suitable subgoals are ones that are reachable: if the planner can choose subgoals such that each subsequent subgoal is reachable given the previous subgoal, then it can reach any goal by ensuring the last subgoal is the true goal. If we use a goal-conditioned policy to reach these goals, how can we quantify how reachable these subgoals are?

One natural choice is to use a goal-conditioned value function which, as previously discussed, provides a measure of reachability. In particular, given the current state $\mathbf{s}$, a policy will reach a goal $\mathbf{g}$ after $t$ time steps if and only if $V(\mathbf{s}, \mathbf{g}, t) = 0$. More generally, given $K$ intermediate subgoals $\mathbf{g}_{1:K} = \mathbf{g}_1, \ldots, \mathbf{g}_K$ and $K+1$ time intervals $t_1, \ldots, t_{K+1}$ that sum to $T_{\max}$, we define the *feasibility vector* as

$$\overrightarrow{\mathbf{V}}(\mathbf{s}, \mathbf{g}_{1:K}, t_{1:K+1}, \mathbf{g}) = \begin{bmatrix} V(\mathbf{s}, \mathbf{g}_1, t_1) \\ V(\mathbf{g}_1, \mathbf{g}_2, t_2) \\ \vdots \\ V(\mathbf{g}_{K-1}, \mathbf{g}_K, t_K) \\ V(\mathbf{g}_K, \mathbf{g}, t_{K+1}) \end{bmatrix}.$$

The feasibility vector provides a quantitative measure of a plan's feasibility: The first element describes how close the policy will reach the first subgoal, $\mathbf{g}_1$, starting from the initial state, $\mathbf{s}$.

The second element describes how close the policy will reach the second subgoal, $\mathbf{g}_2$, starting from the first subgoal, and so on, until the last term measures the reachability to the true goal, $\mathbf{g}$.

To create a feasible plan, we would like each element of this vector to be zero, and so we minimize the norm of the feasibility vector:

$$\mathcal{L}(\mathbf{g}_{1:K}) = ||\overrightarrow{\mathbf{V}}(\mathbf{s}, \mathbf{g}_{1:K}, t_{1:K+1}, \mathbf{g})||. \tag{4.9}$$

In other words, minimizing Equation (4.9) searches for subgoals such that the overall path is feasible and terminates at the true goal. In the next section, we turn to optimizing Equation (4.9) and address issues that arise in high-dimensional state spaces.

#### 4.2.1.2 Optimizing over Images

We consider image-based environments, where the set of states $\mathcal{S}$ is the set of valid image observations in our domain. In image-based environments, solving the optimization in Equation (4.9) presents two problems. First, the optimization variables $\mathbf{g}_{1:K}$ are very high-dimensional – even with 64x64 images and just 3 subgoals, there are over 10,000 dimensions. Second, and perhaps more subtle, the optimization iterates must be constrained to the set of valid image observations $\mathcal{S}$ for the subgoals to correspond to meaningful states. While a plethora of constrained optimization methods exist, they typically require knowing the set of valid states [161] or being able to project onto that set [164]. In image-based domains, the set of states $\mathcal{S}$ is an unknown $r$-dimensional manifold embedded in a higher-dimensional space $\mathbb{R}^N$, for some $N \gg r$ [142] – i.e., the set of valid image observations.

Optimizing Equation (4.9) would be much easier if we could directly optimize over the $r$ dimensions of the underlying representation, since $r \ll N$, and crucially, since we would not have to worry about constraining the planner to an unknown manifold. While we may not know the set $\mathcal{S}$ a priori, we can learn a latent-variable model with a compact latent space to capture it, and then optimize in the latent space of this model. To this end, we use a variational-autoencoder (VAE) [116, 182], which we train with images randomly sampled from our environment.

A VAE consists of an encoder $q_\phi(\mathbf{z} \mid \mathbf{s})$ and decoder $p_\theta(\mathbf{s} \mid \mathbf{z})$. The inference network maps high-dimensional states $\mathbf{s} \in \mathcal{S}$ to a distribution over lower-dimensional latent variables $\mathbf{z}$ for some lower dimensional space $\mathcal{Z}$, while the generative model reverses this mapping. Moreover, the VAE is trained so that the marginal distribution of $\mathcal{Z}$ matches our prior distribution $p_0$, the standard Gaussian.

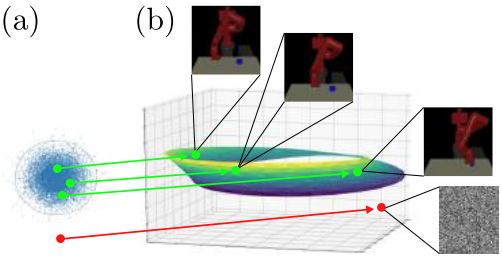



**Figure 4.5:** Optimizing directly over the image manifold (b) is challenging, as it is generally unknown and resides in a high-dimensional space. We optimize over a latent state (a) and use our decoder to generate images. So long as the latent states have high likelihood under the prior (green), they will correspond to realistic images, while latent states with low likelihood (red) will not.

This last property of VAEs is crucial, as it allows us to tractably optimize over the manifold of valid states $\mathcal{S}$. So long as the latent variables have high likelihood under the prior, the corresponding images will remain inside the manifold of valid states, as shown in Figure 4.5. In fact, Dai and Wipf [40] showed that a VAE with a Gaussian prior can always recover the true manifold, making this choice for latent-variable model particularly appealing.

In summary, rather than minimizing Equation (4.9), which requires optimizing over the high-dimensional, unknown space $\mathcal{S}$ we minimize

$$\mathcal{L}_{\text{LEAP}}(\mathbf{z}_{1:K}) = ||\overrightarrow{\mathbf{V}}(\mathbf{s}, \mathbf{z}_{1:K}, t_{1:K+1}, \mathbf{g})||_p - \lambda \sum_{k=1}^{K} \log p(\mathbf{z}_k) \tag{4.10}$$

where

$$\overrightarrow{\mathbf{V}}(\mathbf{s}, \mathbf{z}_{1:K}, t_{1:K+1}, \mathbf{g}) = \begin{bmatrix} V(\mathbf{s}, \psi(\mathbf{z}_1), t_1) \\ V(\psi(\mathbf{z}_1), \psi(\mathbf{z}_2), t_2) \\ \vdots \\ V(\psi(\mathbf{z}_{K-1}), \psi(\mathbf{z}_K), t_K) \\ V(\psi(\mathbf{z}_K), \mathbf{g}, t_{K+1}) \end{bmatrix} \quad \text{and} \quad \psi(\mathbf{z}) = \arg\max_{\mathbf{g}'} p_\theta(\mathbf{g}' \mid \mathbf{z}).$$

This procedure optimizes over latent variables $\mathbf{z}_k$, which are then mapped onto high-dimensional goal states $\mathbf{g}_k$ using the maximum likelihood estimate (MLE) of the decoder $\arg\max_{\mathbf{g}}(\mathbf{g} \mid \mathbf{z})$. In our case, the MLE can be computed in closed form by taking the mean of the decoder. The term summing over $\log p(\mathbf{z}_k)$ penalizes latent variables that have low likelihood under the prior $p$, and $\lambda$ is a hyperparameter that controls the importance of this second term.

While any norm could be used, we used the $\ell_\infty$-norm which forces each element of the feasibility vector to be near zero. We found that the $\ell_\infty$-norm outperformed the $\ell_1$-norm, which only forces the sum of absolute values of elements near zero. [1]

#### 4.2.1.3 Goal-Conditioned Reinforcement Learning

For our goal-conditioned reinforcement learning algorithm, we use temporal difference models (TDMs) [172]. TDMs learn Q functions rather that V functions, and so we compute $V$ by evaluating $Q$ with the action from the deterministic policy: $V(\mathbf{s}, \mathbf{g}, t) = Q(\mathbf{s}, \mathbf{a}, \mathbf{g}, t)|_{\mathbf{a}=\pi(\mathbf{s},\mathbf{g},t)}$. To further improve the efficiency of our method, we can also utilize the same VAE that we use to recover the latent space for planning as a state representation for TDMs. While we could train the reinforcement learning agents from scratch, this can be expensive in terms of sample efficiency as much of the learning will focus on simply learning good convolution filters. We therefore use the pretrained mean-encoder of the VAE as the state encoder for our policy and value function networks, and only train additional fully-connected layers with RL on top of these representations. Details of the architecture are provided in Appendix D.2.3. We show in Section 4.2.2 that our method works without reusing the VAE mean-encoder, and that this parameter reuse primarily helps with increasing the speed of learning.

[1] See **??** D.2.1.1 comparison.

#### 4.2.1.4 Summary of Latent Embeddings for Abstracted Planning

Our overall method is called Latent Embeddings for Abstracted Planning (LEAP) and is summarized in Algorithm 5. We first train a goal-conditioned policy and a variational-autoencoder on randomly collected states. Then at testing time, given a new goal, we choose subgoals by minimizing Equation (4.10). Once the plan is chosen, the first goal $\psi(\mathbf{z}_1)$ is given to the policy. After $t_1$ steps, we repeat this procedure: we produce a plan with $K-1$ (rather than $K$) subgoals, and give the first goal to the policy. In this work, we fix the time intervals to be evenly spaced out (i.e., $t_1 = t_2 \dots t_{K+1} = \lfloor T_{\max}/(K+1) \rfloor$), but additionally optimizing over the time intervals would be a promising future extension.

**Algorithm 5** Latent Embeddings for Abstracted Planning (LEAP)

Train VAE encoder $q_\phi$ and decoder $p_\theta$.
Train TDM policy $\pi$ and value function $V$.
Initialize state, goal, and time: $\mathbf{s}_1 \sim \rho_0$, goal $\mathbf{g} \sim \rho_g$, and $t = 1$.
Assign the last subgoal to the true goal, $\mathbf{g}_{K+1} = \mathbf{g}$
**for** $k$ in $1, \dots, K+1$ **do**
    Optimize Equation (4.10) to choose latent subgoals $\mathbf{z}_k, \dots, \mathbf{z}_K$ using $V$ and $p_\theta$ if $k \leq K$.
    Decode $\mathbf{z}_k$ to obtain goal $\mathbf{g}_k = \psi(\mathbf{z}_k)$.
    **for** $t'$ in $1, \dots, t_k$ **do**
        Sample next action $\mathbf{a}_t$ using goal-conditioned policy $\pi(\cdot \mid \mathbf{s}_t, \mathbf{g}_k, t_k - t')$.
        Execute $\mathbf{a}_t$ and obtain next state $\mathbf{s}_{t+1}$
        Increment the global timer $t \leftarrow t + 1$.
    **end for**
**end for**

### 4.2.2 Experiments

Our experiments study the following two questions: **(1)** How does LEAP compare to model-based methods, which directly predict each time step, and model-free RL, which directly optimizes for the final goal? **(2)** How does the use of a latent state representation and other design decisions impact the performance of LEAP?

#### 4.2.2.1 Vision-based Comparison and Results

We study the first question on two distinct vision-based tasks, each of which requires temporally-extended planning and handling high-dimensional image observations.

The first task, *2D Navigation* requires navigating around a U-shaped wall to reach a goal, as shown in Figure 4.6. The state observation is a top-down image of the environment. We use this task to conduct ablation studies that test how each component of LEAP contributes to

final performance. We also use this environment to generate visualizations that help us better understand how our method uses the goal-conditioned value function to evaluate reachability over images. While visually simple, this task is far from trivial for goal-conditioned and planning methods: a greedy goal-reaching policy that moves directly towards the goal will never reach the goal. The agent must plan a temporally-extended path that moves around the walls, sometimes moving away from the goal. We also use this environment to compare our method with prior work on goal-conditioned and model-based RL.

To evaluate LEAP on a more complex task, we utilize a robotic manipulation simulation of a *Push and Reach* task. This task requires controlling a simulated Sawyer robot to both (1) move a puck to a target location and (2) move its end effector to a target location. This task is more visually complex, and requires more temporally extended reasoning. The initial arm and and puck locations are randomized so that the agent must decide how to reposition the arm to reach around the object, push the object in the desired direction, and then move the arm to the correct location, as shown in Figure 4.6. A common failure case for model-free policies in this setting is to adopt an overly greedy strategy, only moving the arm to the goal while ignoring the puck.

We train all methods on randomly initialized goals and initial states. However, for evaluation, we intentionally select difficult start and goal states to evaluate long-horizon reasoning. For 2D Navigation, we initialize the policy randomly inside the center square and sample a goal from the region directly below the U-shaped wall. This requires initially moving away from the goal to navigate around the wall. For Push and Reach, we evaluate on 5 distinct challenging configurations, each requiring the agent to first plan to move the puck, and then move the arm only once the puck is in its desired location. In one configuration for example, we initialize the hand and puck on opposite sides of the workspace and set goals so that the hand and puck must switch sides.

We compare our method to both model-free methods and model-based methods that plan over learned models. All of our tasks use $T_{\max} = 100$, and LEAP uses CEM to optimize over $K = 3$ subgoals, each of which are 25 time steps apart. We compare directly with model-free TDMs, which we label **TDM-25**. Since the task is evaluated on a horizon of length $T_{\max} = 100$ we also compare to a model-free TDM policy trained for $T_{\max} = 100$, which we label **TDM-100**. We compare to reinforcement learning with imagined goals (**RIG**) [155], a state-of-the-art method for solving image-based goal-conditioned tasks. RIG learns a reward function from images rather than using a pre-determined reward function. We found that providing RIG with the same distance function as our method improves its performance, so we use this stronger variant of RIG to ensure a fair comparison. In addition, we compare to hindsight experiment replay (**HER**) [4] which uses sparse, indicator rewards. Lastly, we compare to probabilistic ensembles with trajectory sampling (PETS) [33], a state-of-the-art model-based RL method. We favorably implemented PETS on the ground-truth low-dimensional state representation and label it **PETS, state**.

The results are shown in Figure 4.6. LEAP significantly outperforms prior work on both tasks, particularly on the harder Push and Reach task. While the TDM used by LEAP (TDM-25) performs poorly by itself, composing it with 3 different subgoals using LEAP

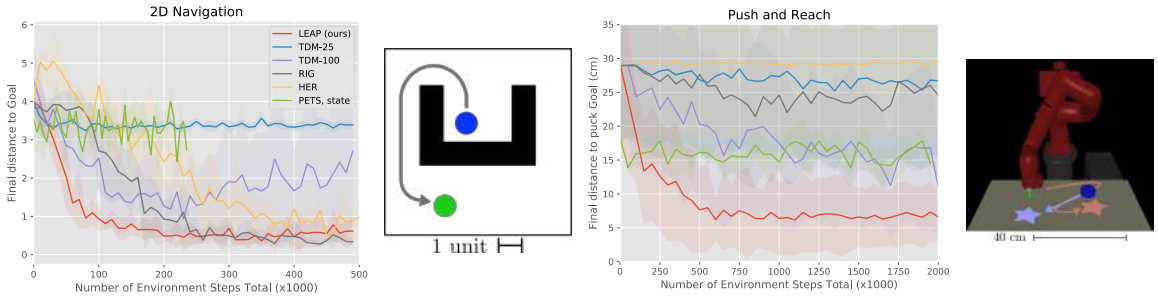



**Figure 4.6:** Comparisons on two vision-based domains that evaluate temporally extended control, with illustrations of the tasks. In 2D Navigation (left), the goal is to navigate around a U-shaped wall to reach the goal. In the Push and Reach manipulation task (right), a robot must first push a puck to a target location (blue star), which may require moving the hand away from the goal hand location, and then move the hand to another location (red star). Curves are averaged over multiple seeds and shaded regions represent one standard deviation. Our method, shown in red, outperforms prior methods on both tasks. On the Push and Reach task, prior methods typically get the hand close to the right location, but perform much worse at moving the puck, indicating an overly greedy strategy, while our approach succeeds at both.

results in much better performance. By 400k environment steps, LEAP already achieves a final puck distance of under 10 cm, while the next best method, TDM-100, requires 5 times as many samples. Details on each task are in Appendix D.2.2, and algorithm implementation details are given in Appendix D.2.3.

We visualize the subgoals chosen by LEAP in Figure 4.7 by decoding the latent subgoals $\mathbf{z}_{t_{1:K}}$ into images with the VAE decoder $p_\theta$. In Push and Reach, these images correspond to natural subgoals for the task. Figure 4.7 also shows a visualization of the value function, which is used by the planner to determine reachability. Note that the value function generally recognizes that the wall is impassable, and makes reasonable predictions for different time horizons. Videos of the final policies and generated subgoals and code for our implementation of LEAP are available on the paper website[2].

#### 4.2.2.2 Planning in Non-Vision-based Environments with Unknown State Spaces

While LEAP was presented in the context of optimizing over images, we also study its utility in non-vision based domains. Specifically, we compare LEAP to prior works on an *Ant Navigation* task, shown in Figure 4.8, where the state-space consists of the quadruped robot's joint angles, joint velocity, and center of mass. While this state space is more compact than images, only certain combinations of state values are actually valid, and the obstacle in the environment is unknown to the agent, meaning that a naïve optimization over the state space can easily result in invalid states (e.g., putting the robot inside an obstacle).

[2] https://sites.google.com/view/goal-planning

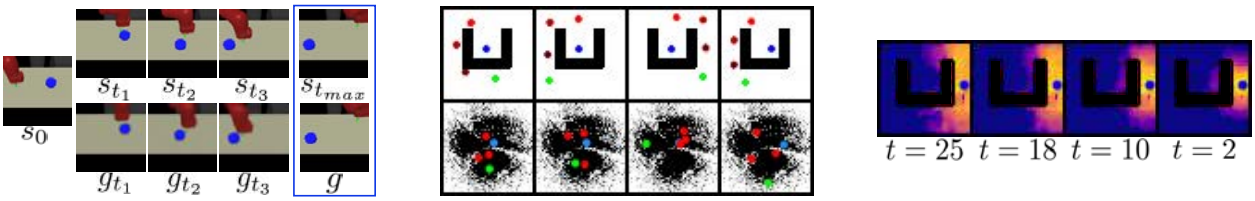



**Figure 4.7:** (Left) Visualization of subgoals reconstructed from the VAE (bottom row), and the actual images seen when reaching those subgoals (top row). Given an initial state $s_0$ and a goal image $\mathbf{g}$, the planner chooses meaningful subgoals: at $\mathbf{g}_{t_1}$, it moves towards the puck, at $\mathbf{g}_{t_2}$ it begins pushing the puck, and at $\mathbf{g}_{t_3}$ it completes the pushing motion before moving to the goal hand position at $\mathbf{g}$. (Middle) The top row shows the image subgoals superimposed on one another. The blue circle is the starting position, the green circle is the target position, and the intermediate circles show the progression of subgoals (bright red is $\mathbf{g}_{t_1}$, brown is $\mathbf{g}_{t_3}$). The colored circles show the subgoals in the latent space (bottom row) for the two most active VAE latent dimensions, as well as samples from the VAE aggregate posterior [144]. (Right) Heatmap of the value function $V(\mathbf{s}, \mathbf{g}, t)$, with each column showing a different time horizon $t$ for a fixed state $\mathbf{s}$. Warmer colors show higher value. Each image indicates the value function for all possible goals $\mathbf{g}$. As the time horizon decreases, the value function recognizes that it can only reach nearby goals.

This task has a significantly longer horizon of $T_{\max} = 600$, and LEAP uses CEM to optimize over $K = 11$ subgoals, each of which are 50 time steps apart. As in the vision-based comparisons, we compare with model-free TDMs, for the short-horizon setting (**TDM-50**) which LEAP is built on top of, and the long-horizon setting (**TDM-600**). In addition to **HER**, we compare to a variant of HER that uses the same rewards and relabeling strategy as RIG, which we label **HER+**. We exclude the PETS baseline, as it has been unable to solve long-horizon tasks such as ours. In this section, we add a comparison to hierarchical reinforcement learning with off-policy correction (**HIRO**) [152], a hierarchical method for state-based goals. We evaluate all baselines on a challenging configuration of the task in which the ant must navigate from one corner of the maze to the other side, by going around a long wall. The desired behavior will result in large negative rewards during the trajectory, but will result in an optimal final state. We see that in Figure 4.8, LEAP is the only method that successfully navigates the ant to the goal. HIRO, HER, HER+ don't attempt to go around the wall at all, as doing so will result in a large sum of negative rewards. TDM-50 has a short horizon that results in greedy behavior, while TDM-600 fails to learn due to temporal sparsity of the reward.

#### 4.2.2.3 Ablation Study

We analyze the importance of planning in the latent space, as opposed to image space, on the navigation task. For comparison, we implement a planner that directly optimizes over image subgoals (i.e., in pixel space). We also study the importance of reusing the pretrained VAE encoder by replicating the experiments with the RL networks trained from scratch.

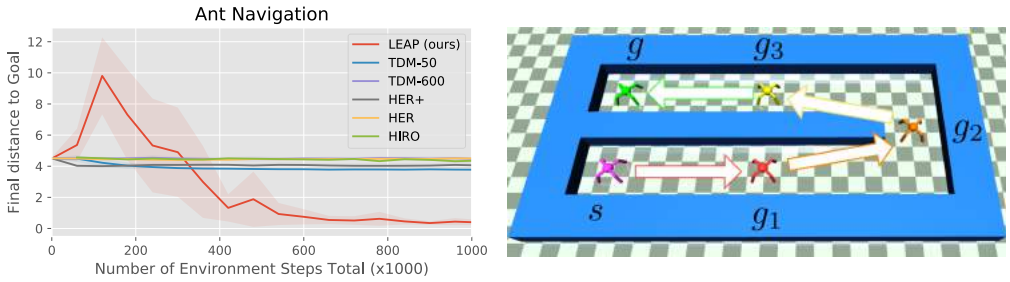



**Figure 4.8:** In the Ant Navigation task, the ant must move around the long wall, which will incur large negative rewards during the trajectory, but will result in an optimal final state. We illustrate the task, with the purple ant showing the starting state and the green ant showing the goal. We use 3 subgoals here for illustration. Our method (shown in red in the plot) is the only method that successfully navigates the ant to the goal.

We see in Figure 4.9 that a model that does not reuse the VAE encoder does succeed, but takes much longer. More importantly, planning over latent states achieves dramatically better performance than planning over raw images. Figure 4.9 also shows the intermediate subgoals outputted by our optimizer when optimizing over images. While these subgoals may have high value according to Equation (4.9), they clearly do not correspond to valid state observations, indicating that the planner is exploiting the value function by choosing images far outside the manifold of valid states.

We include further ablations in Appendix D.2.1, in which we study the sensitivity of $\lambda$ in Equation (4.10) (**??** D.2.1.3), the choice of norm (**??** D.2.1.1), and the choice of optimizer (**??** D.2.1.2). The results show that LEAP works well for a wide range of $\lambda$, that $\ell_\infty$-norm performs better, and that CEM consistently outperforms gradient-based optimizers, both in terms of optimizer loss and policy performance.

## 4.3 Conclusion

In this chapter, we presented a way to reuse goal-directed skills by deriving a connection between model-based and model-free reinforcement learning. We presented a novel RL algorithm that exploits this connection to greatly improve on the sample efficiency of state-of-the-art model-free deep RL algorithms. Our temporal difference models can be viewed both as goal-conditioned value functions and implicit dynamics models, which enables them to be trained efficiently on off-policy data while still minimizing the effects of model bias. As a result, they achieve asymptotic performance that compares favorably with model-free algorithms, but with a sample complexity that is comparable to purely model-based methods.

We also presented LEAP, a method for extending TDMs to solve temporally extended tasks with high-dimensional state observations, such as images. The key idea in LEAP is to form *temporal* abstractions by using goal-reaching policies to evaluate reachability, and *state*

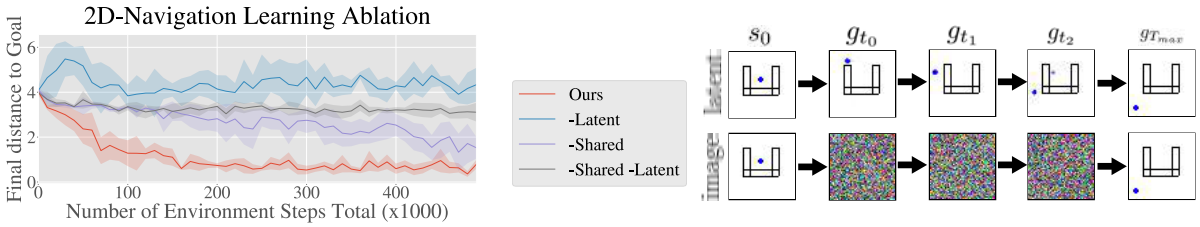



**Figure 4.9:** (Left) Ablative studies on 2D Navigation. We keep all components of LEAP the same but replace optimizing over the latent space with optimizing over the image space (-latent). We separately train the RL methods from scratch rather than reusing the VAE mean encoder (-shared), and also test both ablations together (-latent, shared). We see that sharing the encoder weights with the RL policy results in faster learning, and that optimizing over the latent space is critical for success of the method. (Right) Visualization of the subgoals generated when optimizing over the latent space and decoding the image (top) and when optimizing over the images directly (bottom). The goals generated when planning in image space are not meaningful, which explains the poor performance of "-latent" shown in (Left).

abstractions by using representation learning to provide a convenient state representation for planning. By planning over states in a learned latent space and using these planned states as subgoals for goal-conditioned policies, LEAP can solve tasks that are difficult to solve with conventional model-free goal-reaching policies, while avoiding the challenges of modeling low-level observations associated with fully model-based methods.

Model-based methods represent one way to extend the capabilities of goal-conditioned policies, and in the next chapter, we discuss how we can extend the training methods of goal-conditioned policies to other problem formulations.

# Chapter 5

# Extensions to Non-Goal-Reaching Tasks

The methods presented thus far have focused on goal-reaching tasks. However, goal-reaching cannot represent many of the behaviors we might actually want a versatile robotic system to perform, since it can only represent behaviors that involve reaching individual states. For example, for a robot packing items into a box, the task is defined by the position of the items relative to the box, rather than their absolute locations in space, and therefore does not correspond to a single state configuration. In this chapter, we explore how general-purpose robotic policies can be acquired by conditioning policies on more general task representations. Many of the techniques we have discussed are applicable to any contextual policy, and below we present two examples of how we can train contextual policies to solve various tasks.

## 5.1 Distribution-Conditioned Reinforcement Learning

To make it possible to learn general-purpose policies that can perform any task, we first consider conditioning a policy on a full *distribution* over goal states. Rather than reaching a specific state, a policy must learn to reach states that have high likelihood under the provided distribution, which may specify various covariance relationships (e.g., as shown in Figure 5.1, that the position of the items should covary with the position of the box). In fact, we show that, because optimal policies are invariant to additive factors in reward functions, arbitrary goal distributions can represent *any* state-dependent reward function, and therefore any task. Choosing a specific distribution class provides a natural mechanism to control the expressivity of the policy. We may choose a small distribution class to narrow the range of tasks and make learning easier, or we may choose a large distribution class to expand the expressiveness of the policy.

Our experiments demonstrate that distribution-conditioned policies can be trained efficiently by sharing data from a variety of tasks and relabeling the goal distribution parameters, where each distribution corresponds to a different task reward. Lastly, while the distribution

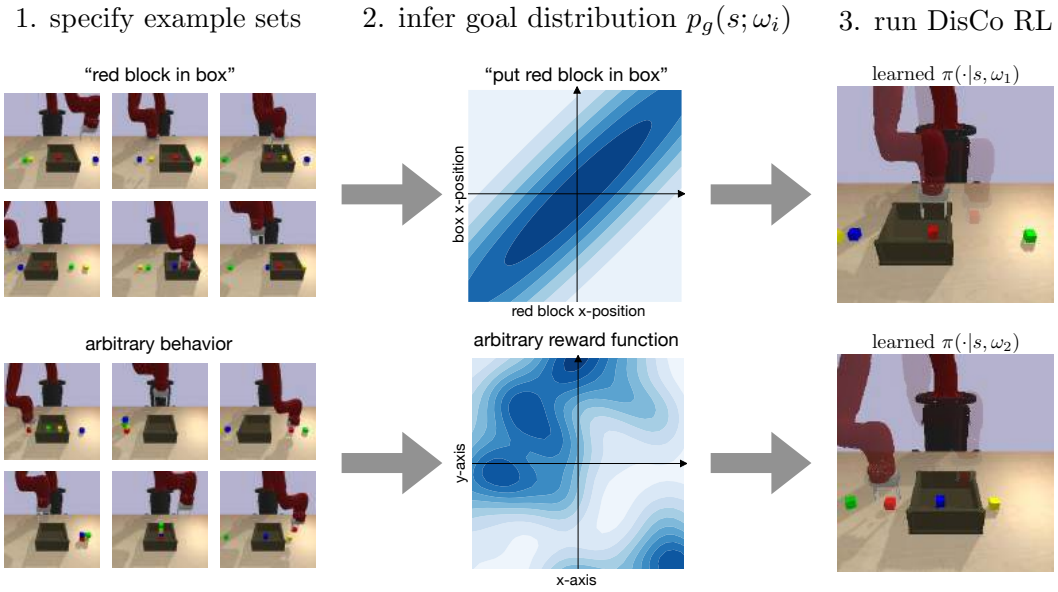



**Figure 5.1:** We infer the distribution parameters $\omega$ from data and pass it to a DisCo policy. Distribution-conditioned RL can express a broad range of tasks, from defining relationships between different state components (top) to more arbitrary behavior (bottom).

parameters can be provided manually to specify tasks, we also present two ways to infer these distribution parameters directly from data.

The main contribution of this section is DisCo RL, an algorithm for learning distribution-conditioned policies. To learn efficiently, DisCo RL uses off-policy training and a novel distribution relabeling scheme. We evaluate on robot manipulation tasks in which a single policy must solve multiple tasks that cannot be expressed as reaching different goal states. We find that conditioning the policies on goal distributions results in significantly faster learning than solving each task individually, enabling policies to acquire a broader range of tasks than goal-conditioned methods.

### 5.1.1 Contextual Markov Decision Processes

A contextual MDP augments an MDP with a context space $\mathcal{C}$, and both the reward and the policy [1] are conditioned on a context $c \in \mathcal{C}$ for the entire episode. Contextual MDPs allow us to easily model policies that accomplish much different tasks, or, equivalently, maximize different rewards, at each episode. The design of the context space and contextual reward function determines the behaviors that these policies can learn. For example, when the context is a goal state, we recover the goal-conditioned reinforcement learning formulation

[1] Some work assumes the dynamics $p$ depend on the context [87]. We assume that the dynamics are the same across all contexts, though our method could also be used with context-dependent dynamics.

described in Section 2.1. If we want to train general-purpose policies that can perform arbitrary tasks, is it possible to design a contextual MDP that captures the set of all rewards functions? In theory, the context space $\mathcal{C}$ could be the set of all reward functions. But in practice, it is unclear how to condition policies on reward functions, since most learning algorithms are not well suited to take functions as inputs. In the next section, we present a promising context space: the space of goal distributions.

### 5.1.2 Distribution-Conditioned Reinforcement Learning

In this section, we show how conditioning policies on a goal distribution results in a MDP that can capture any set of reward functions. Each distribution represents a different reward function, and so choosing a distribution class provides a natural mechanism to choose the expressivity of the contextual policy. We then present distribution-conditioned reinforcement learning (DisCo RL), an off-policy algorithm for training policies conditioned on parametric representation of distributions, and discuss the specific representation that we use.

#### 5.1.2.1 Generality of Goal Distributions

We assume that the goal distribution is in a parametric family, with parameter space $\Omega$, and augment the MDP state space with the goal distribution, as in $\mathcal{S}' = \mathcal{S} \times \Omega$. At the beginning of each episode, a parameter $\omega \in \Omega$ is sampled from some parameter distribution $p_\omega$. The parameter $\omega$ defines the goal distribution $p_g(\mathbf{s}; \omega) : \mathcal{S} \mapsto \mathbb{R}_+$ over the state space. The policy is conditioned on this parameter, and is given by $\pi(\cdot \mid \mathbf{s}, \omega)$. The objective of a distribution-conditioned (DisCo) policy is to reach states that have high log-likelihood under the goal distribution, which can be expressed as

$$\max_\pi \mathbb{E}_{\boldsymbol{\tau} \sim \pi(\cdot|\mathbf{s},\omega)} \left[ \sum_t \gamma^t \log p_g(\mathbf{s}_t; \omega) \right]. \tag{5.1}$$

This formulation can express arbitrarily complex distributions and tasks, as we illustrate in Figure 5.1. More formally:

**Remark 1.** *Any reward maximization problem can be equivalently written as maximizing the log-likelihood under a goal distribution (Equation (5.1)), up to a constant factor.*

This statement is true because, for any reward function of the form $r(\mathbf{s})$, we can define a distribution $p_g(\mathbf{s}) \propto e^{r(\mathbf{s})}$, from which we can conclude that maximizing $\log p_g(\mathbf{s})$ is equivalent to maximizing $r(\mathbf{s})$, up to the constant normalizing factor in the denominator. If the reward function depends on the action, $r(\mathbf{s}, \mathbf{a})$, we can modify the MDP and append the previous action to the state $\bar{\mathbf{s}} = [\mathbf{s}, \mathbf{a}]$, reducing it to another MDP with a reward function of the form $r(\bar{\mathbf{s}})$.

Of course, while any reward can be expressed as the log-likelihood of a goal distribution, a specific fixed parameterization $p_g(\mathbf{s}; \omega)$ may not by itself be able to express any reward.

In other words, choosing the distribution parameterization is equivalent to choosing the set of reward functions that the conditional policy can maximize. As we discuss in the next section, we can trade-off expressivity and ease of learning by choosing an appropriate goal distribution family.

#### 5.1.2.2 Goal Distribution Parameterization

Different distribution classes represent different types of reward functions. To explore the different capabilities afforded by different distributions, we study three families of distributions.

**Gaussian distribution** A simple class of distributions is the family of multivariate Gaussian. Given a state space in $\mathbb{R}^n$, the distribution parameters consists of two components, $\omega = (\mu, \Sigma)$, where $\mu \in \mathbb{R}^n$ is the mean vector and $\Sigma \in \mathbb{R}^{n \times n}$ is the covariance matrix. When inferring the distribution parameters with data, we regularize the $\Sigma^{-1}$ matrix by thresholding absolute values below $\epsilon = 0.25$ to zero. With these parameters, the reward from Equation (5.1) is given by $r(\mathbf{s}; \omega) = -0.5(\mathbf{s} - \mu)^T \Sigma^{-1} (\mathbf{s} - \mu)$, where we have dropped constant terms that do not depend on the state $\mathbf{s}$. This simple parameterization can express a large number of reward functions. Using this parameterization, the weight of individual state dimensions depend on the values along the diagonal of the covariance matrix. By using off-diagonal covariance values, this parameterization also captures the set of tasks in which state components need to covary, such as when the one object must be placed near another one, regardless of the exact location of those objects (see the top half of Figure 5.1).

**Gaussian mixture model** A more expressive class of distributions that we study is the Gaussian mixture model with 4 modes, which can represent multi-modal tasks. The parameters are the mean and covariance of each Gaussian and the weight assigned to each Gaussian distribution. The reward is given by the log-likelihood of a state.

**Latent variable model** To study an even more express class of distribution, we consider a class of distributions parameterized by neural networks. Distributions parameterized by neural networks can be extremely expressive [48, 212], but distributions based on neural networks often have millions of parameters [48, 117].

To obtain an expressive yet compact parameterization, we consider non-linear reparameterizations of the original state space. Specifically, we model a distribution over the state space using a Gaussian variational auto-encoder (VAE) [116, 182]. Gaussian VAEs model a set of observations using a latent-variable model of the form $p(\mathbf{s}) = \int_{\mathcal{Z}} p(\mathbf{z}) p_\theta(\mathbf{s} \mid \mathbf{z}) d\mathbf{z}$, where $\mathbf{z} \in \mathcal{Z} = \mathbb{R}^{d_z}$ are latent variables with dimension $d_z$. The distribution $p_\theta$ is a learned generative model or "decoder" and $p(z)$ is a standard multivariate Gaussian distribution in $\mathbb{R}^{d_z}$. A Gaussian VAE also learns a posterior distribution or "encoder" that maps states $\mathbf{s}$ onto Gaussian distributions in a latent space, given by $q_\phi(\mathbf{z}; \mathbf{s})$. We refer readers to Doersch [49] for a detailed explanation of VAEs.

Gaussian VAEs are explicitly trained so that Gaussian distributions in a latent space define distributions over the state space. Therefore, we represent a distribution over the state space with the mean $\mu_z \in \mathbb{R}^{d_z}$ and variances $\sigma_z \in \mathbb{R}^{d_z}$ of a diagonal Gaussian distribution *in the learned, latent space*, which we write as $\mathcal{N}(\mathbf{z}; \mu_z, \sigma_z)$. In other words, the parameters $\omega = (\mu_z, \sigma_z)$ define the following distribution over the states:

$$p_g(\mathbf{s}; \omega) = \int_{\mathcal{Z}} \mathcal{N}(\mathbf{z}; \mu_z, \sigma_z) p_\theta(\mathbf{s} \mid \mathbf{z}) d\mathbf{z}, \tag{5.2}$$

where $p_\theta(\mathbf{s} \mid \mathbf{z})$ is the generative model from the VAE. Because $\mathbf{z}$ resides in a learned latent space, the set of reward functions that can be expressed with Equation (5.2) includes arbitrary non-linear transformations of $\mathbf{s}$.

### 5.1.3 Learning Goal Distributions and Policies

Given any of the distribution classes mentioned above, we now consider how to obtain a specific goal distribution parameter $\omega$ and train a policy to maximize Equation (5.1). We start with obtaining goal distribution parameters $\omega$. While a user can manually select the goal distribution parameters $\omega$, this requires a degree of user insight, which can be costly or practically impossible if using the latent representation in Equation (5.2). We discuss two automated alternatives for obtaining the goal distribution parameters $\omega$.

#### 5.1.3.1 Inferring Distributions from Examples

One simple and practical way to specify a goal distribution is to provide $K$ example observations $\{\mathbf{s}_k\}_{k=1}^K$ in which the task is successfully completed. This supervision can be easier to provide than full demonstrations, which not only specify the task but also must show how to solve the task through a sequence of states $(\mathbf{s}_1, \mathbf{s}_2, \dots)$ or states and actions $(\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \dots)$. Given the example observations, we describe a way to infer the goal parameter based on the parameterization.

If $\omega$ represents the parameters of a distribution in the state space, we learn a goal distribution via maximum likelihood estimation (MLE), as in

$$\omega^* = \arg\max_{\omega \in \Omega} \sum_{k=1}^{K} \log p_g(\mathbf{s}_k; \omega), \tag{5.3}$$

and condition the policy on the resulting parameter $\omega^*$. If $\omega$ represents the parameters of a distribution in the latent space, we need a Gaussian distribution in the latent space that places high likelihood on all of the states in $\mathcal{D}_{\text{subtask}}$. We obtain such a distribution by finding a latent distribution that minimizes the KL divergence to the mixture of posteriors $\frac{1}{K}\sum_k q_\phi(\omega; \mathbf{s}_k)$. Specifically, we solve the problem

$$\omega^* = \arg\min_{\omega \in \Omega} D_{\text{KL}} \left( \frac{1}{K} \sum_k q_\phi(\mathbf{z}; \mathbf{s}_k) \middle\| p(\mathbf{z}; \omega) \right), \tag{5.4}$$

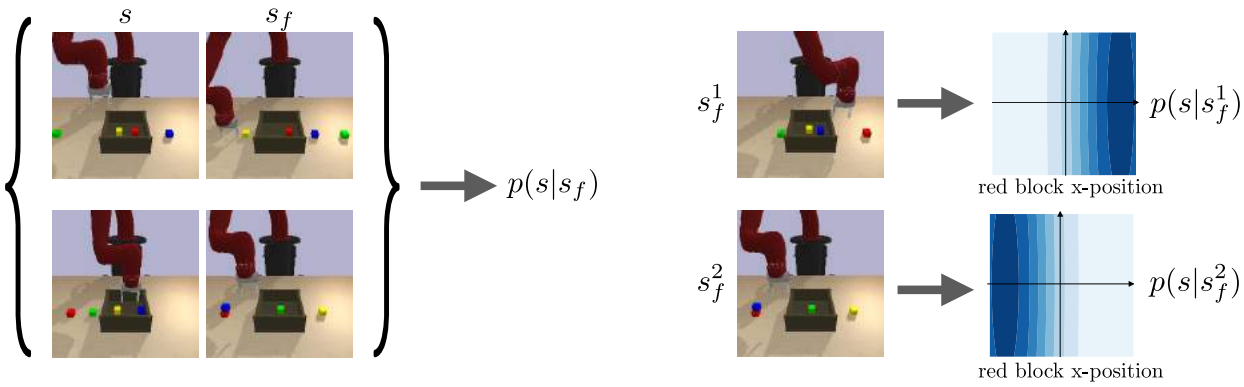



**Figure 5.2:** A robot must arrange objects into a configuration $\mathbf{s}_f$ that changes from episode to episode. This task consists of multiple sub-tasks, such as first moving the red object to the correct location. Given a final state $\mathbf{s}_f$, there exists a distribution of intermediate states $\mathbf{s}$ in which the first sub-task is completed. We use pairs of states $\mathbf{s}, \mathbf{s}_f$ to learn a conditional distribution that defines the first sub-task given the final task, $p(\mathbf{s} \mid \mathbf{s}_f)$.

where $\Omega$ is the set of all means and diagonal covariance matrices in $\mathbb{R}^{d_z}$. The solution to Equation (5.4) can be computed in closed form using moment matching [147].

#### 5.1.3.2 Dynamically Generating Distributions via Conditioning

We also study a different use case for training a DisCo policy: automatically decomposing long-horizon tasks into sub-tasks. Many complex tasks lend themselves to such a decomposition, and learning each of these sub-tasks can be significantly faster than directly solving the main task. For example, a robot that must arrange a table can divide this task into placing one object to a desired location before moving on to the next object. To automatically decompose a task, we need to convert a "final task," represented as $\mathcal{T}$, into $M$ different sub-tasks, where $M$ is the number of sub-tasks needed to accomplish the final task. Moreover, rather than training a separate policy for each sub-task, we would like to train a single policy that can accomplish all of these sub-tasks. How can we convert a task parameter $\mathcal{T}$ into different sub-tasks, all of which can be accomplished by a single policy?

We can address this question by training a DisCo policy. A task represented by parameters $\mathcal{T}$ can be decomposed into a sequence of sub-tasks, each of which in turn is represented by a goal distribution. We accomplish this by learning *conditional distributions*, $p_g^i(\mathbf{s} \mid \mathcal{T})$: conditioned on some final task $\mathcal{T}$, the conditional distribution $p_g^i(\mathbf{s} \mid \mathcal{T})$ is a distribution over desired states for sub-task $i$.

To obtain a conditional distribution $p_g^i(\mathbf{s} \mid \mathcal{T})$ for sub-task $i$, we assume access to tuples $\mathcal{D}^i_{\text{subtask}} = \{(\mathbf{s}^{(k)}, \mathcal{T}^{(k)})\}_{k=1}^K$, where $\mathbf{s}^{(k)}$ is an example state in which sub-task $i$ is accomplished for solving task $\mathcal{T}^{(k)}$. See Figure 5.2 for a visualization. Our experiments test the setting where final tasks are represented by a final state $\mathbf{s}_f$ that we want the robot to reach, meaning that $\mathcal{T} = \mathbf{s}_f$. We note that one can train a goal-conditioned policy to reach this final state $\mathbf{s}_f$

directly, but, as we will discuss in Section 5.1.4, this decomposition significantly accelerates learning by exploiting access to the pairs of states.

For each $i$, and dropping the dependence on $i$ for clarify, we learn a conditional distribution, by fitting a joint Gaussian distribution, denoted by $p_{\mathbf{s},\mathbf{s}_f}(\mathbf{s}, \mathbf{s}_f; \mu, \Sigma)$, to these pairs of states using MLE, as in

$$\mu^*, \Sigma^* = \arg\max_{\mu,\Sigma} \sum_{k=1}^{K} \log p_{\mathbf{s},\mathbf{s}_f}(\mathbf{s}^{(k)}, \mathbf{s}_f^{(k)}; \mu, \Sigma). \tag{5.5}$$

This procedure requires up-front supervision for each sub-task during training time, but at test time, given a desired final state $\mathbf{s}_f$, we compute the parameters of the Gaussian conditional distribution $p_g(\mathbf{s} \mid \mathbf{s}_f; \mu^*, \Sigma^*)$ in closed form. Specifically, the conditional parameters are given by

$$\begin{aligned} \bar{\mu} &= \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(\mathbf{s}_f - \mu_2) \\ \bar{\Sigma} &= \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}, \end{aligned} \tag{5.6}$$

where $\mu_1$ and $\mu_2$ represents the first and second half of $\mu^*$, and similarly for the covariance terms. To summarize, we learn $\mu^*$ and $\Sigma^*$ with Equation (5.5) and then use Equation (5.6) to automatically transform a final task $\mathcal{T} = \mathbf{s}_f$ into a goal distribution $\omega = (\bar{\mu}, \bar{\Sigma})$ which we give to a DisCo policy. Having presented two ways to obtain distributions, we now turn to learning DisCo RL policies.

#### 5.1.3.3 Learning Distribution-Conditioned Policies

In this section, we discuss how to optimize Equation (5.1) using an off-policy TD algorithm. As discussed in Section 2.1, TD algorithms require tuples of state, action, next state, and reward, denoted by $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$. To collect this data, we condition a policy on a goal distribution parameter $\omega$, collect a trajectory with the policy $\boldsymbol{\tau} = [\mathbf{s}_0, \mathbf{a}_0, \cdots]$, and then store the trajectory and distribution parameter into a replay buffer [149], denoted as $\mathcal{R}$. We then sample data from this replay buffer to train our policy using an off-policy TD algorithm. We use soft actor-critic as our RL algorithm [83], though in theory any off-policy algorithm could be used. Because TD algorithms are off-policy, we propose to reuse data collected by a policy conditioned on one goal distribution $\omega$ to learn about how a policy should behave under another goal distribution $\omega'$. In particular, given a state $\mathbf{s}$ sampled from a policy that was conditioned on some goal distribution parameters $\omega$, we occasionally relabel the goal distribution with an alternative goal distribution $\omega' = RS(\mathbf{s}, \omega)$ for training, where $RS$ is some relabeling strategy. For relabeling, given a state $\mathbf{s}$ and existing parameters $\omega = (\mu, \Sigma)$, we would like to provide a strong learning signal by creating a distribution parameter that gives high reward to an achieved state. We used a simple strategy that we found worked well: we replace the mean with the state vector $\mathbf{s}$ and randomly re-sampling the covariance from the set of observed covariances as in $RS(\mathbf{s}, (\mu, \Sigma)) = (\mathbf{s}, \Sigma')$, where $\Sigma'$ is sampled uniformly from the replay buffer. This relabeling is similar to relabeling methods used in goal-conditioned

**Algorithm 6** (Conditional) Distribution-Conditioned RL

**Require:** Policy $\pi_\theta$, $Q$-function $Q_\phi$, TD algorithm $\mathcal{A}$, relabeling strategy $RS$, exploration parameter distribution $p_\omega$, replay buffer $\mathcal{R}$.

Compute $\omega$ using Equation (4) or (5) (if unconditional).
**for** $0, \ldots, N_{\text{episode}} - 1$ episodes **do**
  Sample $\mathbf{s}_f$ and compute $\omega$ with Equation (5.6).
  Sample trajectory from environment $\boldsymbol{\tau} \sim \pi(\cdot|\mathbf{s}; \omega)$ and store tuple $(\boldsymbol{\tau}, \omega)$ in replay buffer $\mathcal{R}$.
  **for** $0, \ldots, N_{\text{updates}} - 1$ steps **do**
    Sample trajectory and parameter $(\boldsymbol{\tau}, \omega) \sim \mathcal{R}$.
    Sample transition tuple $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ and a future state $\mathbf{s}_h$ from $\boldsymbol{\tau}$, where $t < h$.
    With probability $p_{\text{relabel}}$, relabel $\omega \leftarrow RS(\mathbf{s}_h, \omega)$.
    Compute reward $r = \log p_g(\mathbf{s}_t; \omega)$ and augment states $\hat{\mathbf{s}}_t \leftarrow [\mathbf{s}_t; \omega]$, $\hat{\mathbf{s}}_{t+1} \leftarrow [\mathbf{s}_{t+1}; \omega]$.
    Update $Q_\phi$ and $\pi_\theta$ using $\mathcal{A}$ and $(\hat{\mathbf{s}}_t; \mathbf{a}_t, \hat{\mathbf{s}}_{t+1}, r)$.
  **end for**
**end for**

reinforcement learning [104, 4, 177, 172, 152, 171, 155, 60], but applied to goal distributions rather than individual goal states.

We call the method *DisCo RL* when learning a distribution from examples and *Conditional DisCo RL* when learning a conditional distribution. We summarize both in Algorithm 6.

### 5.1.4 Experiments

Our experiments study the following questions: **(1)** How does DisCo RL with a learned distribution compare to prior work that also uses successful states for computing rewards? **(2)** Can we apply Conditional DisCo RL to solve long-horizon tasks that are decomposed into shorter sub-tasks? **(3)** How do DisCo policies perform when conditioned on goal distributions that were never used for data-collection? The first two questions evaluate the variants of DisCo RL presented in Section 5.1.3, and the third question studies how well the method can generalize to test time task specifications. We also include ablations that study the importance of the relabeling strategy presented in Section 5.1.3.3. Videos of our method and baselines are available on the paper website.[2]

We study these questions in three simulated manipulation environments of varying complexity, shown in Figure 5.3. We first consider a simple two-dimensional "Flat World" environment in which an agent can pick up and place objects at various locations. The second environment contains a Sawyer robot, a rectangular box, and four blocks, which the robot must learn to manipulate. The agent controls the velocity of the end effector and gripper, and the arm is restricted to move in a 2D plane perpendicular to the table's surface. Lastly, we use an IKEA furniture assembly environment from Lee et al. [131]. An agent controls

[2] https://sites.google.com/view/disco-rl

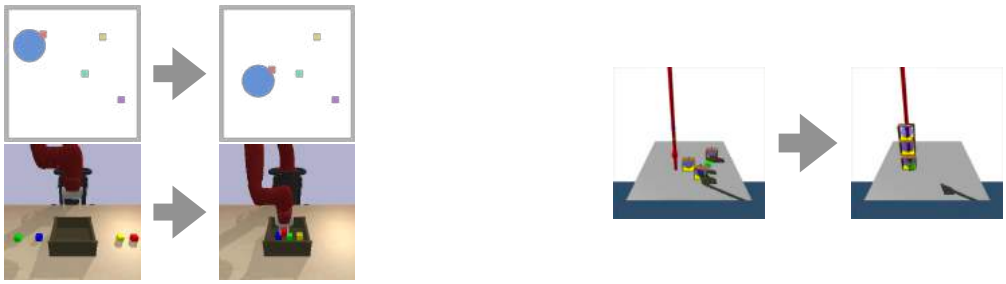


**Figure 5.3:** Illustrations of the experimental domains, in which a policy must (left top) use the blue cursor to move objects to different locations, (left bottom) control a Sawyer arm to move cubes into and out of a box, and (right) attach shelves to a pole using a cursor.

the velocity of a cursor that can lift and place 3 shelves onto a pole. Shelves are connected automatically when they are within a certain distance of the cursor or pole. The states comprise the Cartesian position of all relevant objects and, for the Sawyer task, the gripper state. For the Sawyer environment, we also consider an image-based version which uses a 48x48 RGB image for the state. To accelerate learning on this image environment, we follow past work on image-based RL [155] and encode images $s$ with the mean of the VAE encoder $q_\phi(\mathbf{z}; \mathbf{s})$ for the RL algorithm and pretrain the VAE on 5000 randomly collected images. All plots show mean and standard deviation across 5 seeds, as well as optimal performance with a dashed line.

#### 5.1.4.1 Learning from Examples

Our first set of experiments evaluates how well DisCo RL performs when learning distribution parameters from a fixed set of examples, as described in Section 5.1.3.1. We study the generality of DisCo RL across all three parameterizations from Section 5.1.2.2 on different environments: First, we evaluate DisCo RL with a Gaussian model learned via Equation (5.3) on the Sawyer environment, in which the policy must move the red object into the box while ignoring the remaining three "distractor" objects. Second, we evaluate DisCo RL with a GMM model also learned via Equation (5.3) on the Flat World environment, where the agent must move a specific object to any one of four different locations. Lastly, we evaluate DisCo RL with a latent variable model learned via Equation (5.4) on an image-based version of the Sawyer environment, where the policy must move the hand to a fixed location and ignore visual distractions. We note that this last experiment is done completely from images, and so manually specifying the parameters of a Gaussian in image-space would be impractical. The experiments used between $K = 30$ to 50 examples each for learning the goal distribution parameters. We report the normalized final distance: a value 1 is no better than a random policy.

We compare to past work that uses example states to learn a reward function. Specifically, we compare to variational inverse control with events (**VICE**) [70], which trains a success classifier to predict the user-provided example states as positive and replay buffer states as negative, and then uses the log-likelihood of the classifier as a reward. We also include

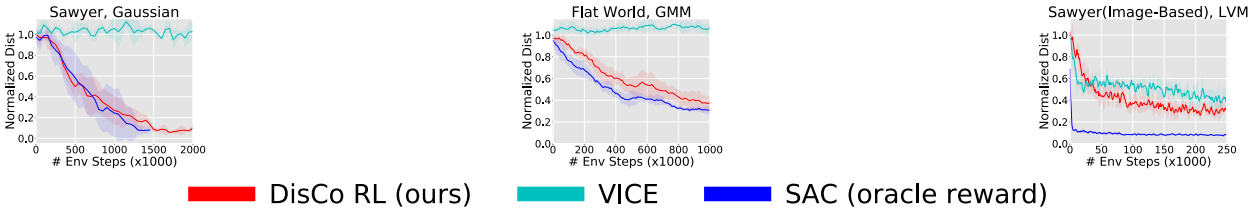



**Figure 5.4:** (Lower is better.) Learning curve showing normalized distance versus environment steps of various method. DisCo RL uses a (left) Gaussian model, (middle) Gaussian mixture model, or (right) latent-variant model on their respective tasks. DisCo RL with a learned goal distribution consistently outperforms VICE and obtains a final performance similar to using oracle rewards.

an oracle labeled **SAC (oracle reward)** which uses the ground truth reward. Since VICE requires training a separate classifier for each task, these experiments only test these methods on a single task. Note that a single DisCo RL policy can solve multiple tasks by conditioning on different parameter distributions, as we will study in the next section.

We see in Figure 5.4 that DisCo RL often matches the performance of using an oracle reward and consistently outperforms VICE. VICE often failed to learn, possibly because the method was developed using hundreds to thousands of examples, where as we only provided 30 to 50 examples.

#### 5.1.4.2 Conditional Distributions for Sub-Task Decomposition

The next experiments study how Conditional DisCo RL can automatically decompose a complex task into easier sub-tasks. We design tasks that require reaching a desired final state $\mathbf{s}_f$, but that can be decomposed into smaller sub-tasks. The first task requires controlling the Sawyer robot to move 4 blocks with randomly initialized positions into a box at a fixed location. We design analogous tasks in the IKEA environment (with 3 shelves and a pole) and Flat World environment (with 4 objects). All of these tasks can be split into sub-task that involving moving a single object at a time.

For each object $i = 1, \ldots, M$, we collect an example set $\mathcal{D}^i_{\text{subtask}}$. As described in Section 5.1.3.2, each set $\mathcal{D}^i_{\text{subtask}}$ contains $K = 30$ to 50 pairs of state $(\mathbf{s}, \mathbf{s}_f)$, in which object $i$ is in the same location as in the final desired state $\mathbf{s}_f$, as shown in Figure 5.2. We fit a joint Gaussian to these pairs using Equation (5.5). During exploration and evaluation, we sample an initial state $\mathbf{s}_0$ and final goal state $\mathbf{s}_f$ uniformly from the set of possible states, and condition the policy on $\omega$ given by Equation (5.6). Because the tasks are randomly sampled, the specific task and goal distribution at evaluation are unseen during training, and so this setting tested whether Conditional DisCo RL can generalize to new goal distributions.

For evaluation, we test how well DisCo RL can solve long-horizon tasks by conditioning the policy on each $p^i_g(\mathbf{s} \mid \mathbf{s}_f)$ sequentially for $H/M$ time steps. We report the cumulative number of tasks that were solved, where each task is considered solved when the respective object is within a minimum distance of its target location specified by $\mathbf{s}_f$. For the IKEA environment, we also consider moving the pole to the correct location as a task. We compare

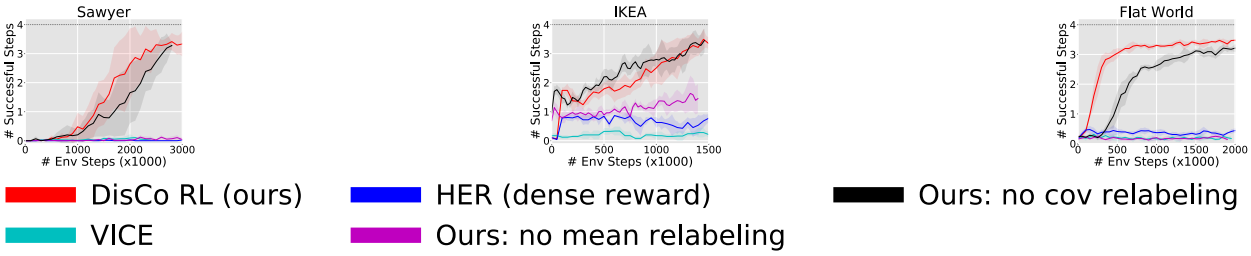



**Figure 5.5:** (Higher is better.) Learning curves showing the number of cumulative tasks completed versus environment steps for the Sawyer (left), IKEA (middle), and Flat World (right) tasks. We see that DisCo RL significantly outperforms HER and VICE, and that relabeling the mean and covariance is important.

to VICE which trains separate classifiers and policies for each sub-task using the examples and also sequentially runs each policies for $H/M$ time steps.

One can train a goal-conditioned policy to reach this final state $\mathbf{s}_f$ directly, and so we compare to hindsight experience replay (**HER**) [4], which attempts to directly reach the final state for $H$ time steps and learns using an oracle dense reward. Since HER does not decompose the task into subtasks, we expect that the learning will be significantly slower since it does not exploit access to the tuples that give examples of a subtask being completed. We see in Figure 5.5 that Conditional DisCo RL significantly outperforms VICE and HER and that Conditional DisCo RL successfully generalizes to new goal distributions.

**Ablations** Lastly, we include ablations that test the importance of relabeling the mean and covariance parameters during training. We see in Figure 5.5 that relabeling both parameters, and particularly the mean, accelerates learning.

Overall, our experiments show that DisCo policies can solve a variety of tasks using goal distributions inferred from data, and can accomplish tasks specified by goal distributions that were not seen during training. In the next section, we discuss another class of contextual policies that can generalize to unseen tasks.

## 5.2 Contextual and Meta-Reinforcement Learning

Another promising class of contextual policies are meta-policies, and in this section we discuss challenges that arise when training such contextual policies and ways to mitigate these challenges. The discussion and lessons here are useful not only for training meta-policies, but also for training contextual policies in general, such as goal-conditioned policies.

Just like goal-conditioned policies, a single meta-policies can solve a large number of tasks. However, rather than being conditioned on a goal state, meta-policies are conditioned on histories of states, actions, and resulting rewards, and must infer the task based on this history. Once this inference procedure is trained, meta-policies can adapt to new tasks with orders of magnitude less data than standard RL [176]. However, the meta-training phase

in these algorithms still requires a large number of online samples, often even more than standard RL, due to the multi-task nature of the meta-learning problem.

Offline reinforcement learning methods, which use only prior experience without active data collection, provide a potential solution to this issue, because a user must only annotate multi-task data with rewards once in the offline dataset, rather than doing so in the inner loop of RL training, and the same offline multi-task data can be reused repeatedly for many training runs. While a few recent works have proposed offline meta-RL algorithms [50, 148], we identify a specific problem when an agent trained with offline meta-RL is tested on a new task: the distributional shift between the behavior policy and the meta-test time exploration policy means that adaptation procedures learned from offline data might not perform well on the (differently distributed) data collected by the exploration policy at meta-test time. In practice, we find that this issue can lead to a large degradation in performance on offline meta-RL tasks. This mismatch in training distribution occurs because meta-RL is never trained on data generated by the exploration policy.

We will discuss how we can address this challenge by collecting additional online data *without* any reward supervision, leading to a semi-supervised offline meta-RL algorithm, as illustrated in Figure 5.6. Online data can be relatively cheap to collect when it does not require reward labels and enables the agent to observe new exploration trajectories. We would like to meta-train on this online data to mitigate the distribution shift that occurs at meta-test time, but meta-training requires reward labels. If an agent can generate its own reward labels for these new states and actions, it can meta-train on this labeled dataset that includes trajectories generated by the learned exploration policy.

Based on this principle, we develop a method called semi-supervised meta actor-critic (SMAC) that uses reward-labeled offline data to bootstrap a semi-supervised meta-reinforcement learning procedure, in which an offline meta-RL agent collects additional online experience without any reward labels. The agent uses the reward supervision from the offline dataset to learn to generate new reward functions, which it uses to autonomously annotate rewards in these otherwise rewardless interactions and meta-train on this new data. We evaluate our method and prior offline meta-RL methods on existing benchmarks [50, 148] with fewer than 400 time steps of reward labels at meta-test time. We find that while standard meta-RL methods perform well at adapting to training tasks, they suffer from data-distribution shifts when they must adapt to new tasks that were not seen during meta-training. In contrast, we find that additional environment interaction greatly improves the meta-test time performance of SMAC, despite the lack of additional reward supervision.

### 5.2.1 Related Works in Meta-Reinforcement Learning

Many prior meta-RL algorithms assume that reward labels are provided with each episode of online interaction [52, 65, 80, 224, 89, 176, 99, 118, 235, 225, 233, 108]. In contrast to these prior methods, our method only requires offline prior data with rewards, and additional online interaction does not require any ground truth reward signal.

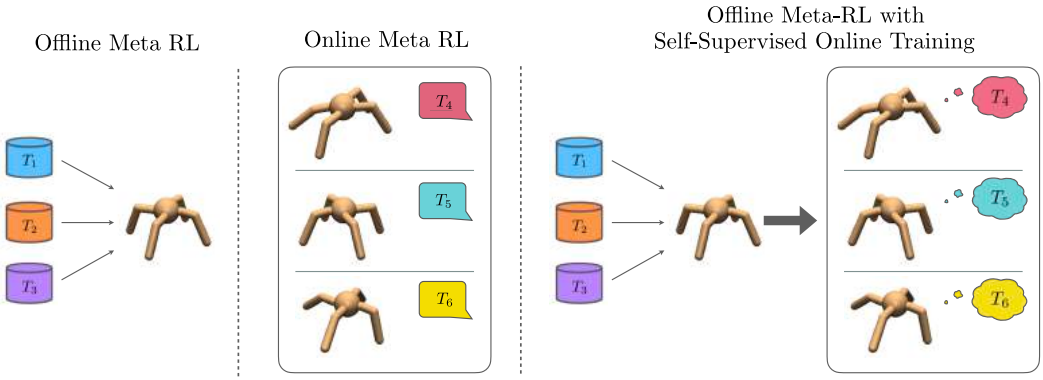



**Figure 5.6:** (left) In offline meta-RL, an agent uses offline data from multiple tasks $T_1, T_2, \ldots$, each with reward labels that must only be provided once. (middle) In online meta-RL, new reward supervision must be provided with every environment interaction. (right) In semi-supervised meta-RL, an agent uses an offline dataset collected once to learn to generate its own reward labels for new, online interactions. Similar to offline meta-RL, reward labels must only be provided once for the offline training, and unlike online meta-RL, the additional environment interactions do not require external reward supervision.

Prior works have also studied other formulations that combine unlabeled and labeled trials. For example, imitation and inverse reinforcement learning methods use offline demonstrations to either learn a reward function [1, 63, 96, 69] or to directly learn a policy [189, 184, 96, 180, 167]. Semi-supervised and positive-unlabeled reward learning [222, 236, 119] methods use reward labels provided for some interactions to train a reward function for RL. However, all of these methods have been studied in the context of a single task. While these methods focus on recovering a policy that maximizes a specific reward function, we focus on meta-learning an RL procedure that can adapt to new reward functions. In other words, we do not focus on recovering a single reward function, because there is no single test time reward or task. Instead, we focus on generating reward labels for meta-training that mitigate the distribution shift between the offline data and online data at test-time.

SMAC uses a context-based adaptation procedure similar to Rakelly et al. [176], which is related to other work on contextual policies, such as goal-conditioned reinforcement learning [104, 190, 4, 172, 34, 217, 168, 155] or successor features [120, 10, 12, 76]. In contrast to these latter works on contextual policies, our meta-learning procedure applies to any RL problem, does not assume that the reward is defined by a single goal state or fixed basis function, and uses offline data to learn to generate rewards.

Our method addresses a similar problem to prior offline meta-RL methods [148, 50], but we show that these approaches generally underperform in low-data regimes, whereas our method addresses the distribution shift problem by using online interactions without requiring additional reward supervision. In our experiments, we found that SMAC greatly improves the performance on both training and held-out tasks. Lastly, SMAC is also related to unsupervised meta-learning methods [79, 101], which annotate data with their own rewards. In contrast to these methods, we assume that there exists an offline dataset with reward

labels that we can use to learn to generate similar rewards.

### 5.2.2 Preliminaries

**Meta-reinforcement learning.** In meta-RL, we assume there is a distribution of tasks $p_{\mathcal{T}}(\cdot)$. A task $\mathcal{T}$ is a Markov decision process (MDP), defined by a tuple $\mathcal{T} = (\mathcal{S}, \mathcal{A}, r, \gamma, p_0, p_d)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $r$ is a reward function, $\gamma$ is a discount factor, $p_0(\mathbf{s}_0)$ is the initial state distribution, and $p_d(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ is the environment dynamics distribution. A replay buffer $\mathcal{D}$ is a set of state, action, reward, next-states tuples, $\mathcal{D} = \{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}_{i=1}^{N_{\text{size}}}$, where all the rewards come from the same task. We will use the letter $\mathbf{h}$ to denote a small replay buffer or "history" and the notation $\mathbf{h} \sim \mathcal{D}$ to denote that a mini-batch $\mathbf{h}$ is sampled from a replay buffer $\mathcal{D}$. We will use the letter $\boldsymbol{\tau}$ to represent a trajectory $\boldsymbol{\tau} = (\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \dots)$ without reward labels.

A meta-episode consists of sampling a task $\mathcal{T} \sim p_{\mathcal{T}}(\cdot)$, collecting $T_{\text{adapt}}$ trajectories with a policy $\pi$, adapting the policy to the task between each trajectory, and measuring the performance on the last trajectory. We write the policy's adaptation procedure as $A_\phi$, parameterized by meta-parameters $\phi$. Between each trajectory, the adaptation procedure transforms the history of interactions $\mathbf{h}$ within the meta-episode into a context $\mathbf{z} = A_\phi(\mathbf{h})$ that summarizes the previous interactions. This context $\mathbf{z} \in \mathcal{Z}$ is then given to the policy $\pi(\mathbf{a}, \mid \mathbf{s}, \mathbf{z})$. The exact representation of $\pi$, $A_\phi$, and $\mathcal{Z}$ depends on the specific meta-RL method used. For example, the context $\mathbf{z}$ can be weights of a neural network [65] outputted by a gradient update, hidden activations outputted by a recurrent neural network [52], or latent variables outputted by a stochastic encoder [176]. Using this notation, the objective in meta-RL is to learn the adaptation parameters $\phi$ and policy parameters $\theta$ to maximize performance on a meta-episode given a new task $\mathcal{T}$ sampled from $p(\mathcal{T})$.

**PEARL.** Since we require an off-policy meta-RL procedure for offline meta-training, we build on probabilistic embeddings for actor-critic RL (PEARL) [176], an online off-policy meta-RL algorithm. In PEARL, $\mathbf{z}$ is a vector and the adaptation procedure $A_\phi$ that maps $\mathbf{h}$ to $\mathbf{z}$ consists of sampling $\mathbf{z}$ from a distribution $\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{h})$. The distribution $q_\phi$ is generated by an encoder network with parameters $\phi$. This encoder is a set-based network that processes all of the tuples in $\mathbf{h} = \{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}_{i=1}^{N_{\text{enc}}}$ in a permutation-invariant manner to produce the mean and variance of a diagonal multivariate Gaussian. The policy is a contextual policy $\pi(\mathbf{a} \mid \mathbf{s}, \mathbf{z})$ conditioned on $\mathbf{z}$ by concatenating $\mathbf{z}$ to the state $\mathbf{s}$.

The policy parameter $\theta$ is trained using soft-actor critic [85] which involves learning a critic, or $Q$-function, $Q_w(\mathbf{s}, \mathbf{a}, \mathbf{z})$, with parameter $w$ that estimates the sum of future discounted rewards conditioned on the current state, action, and context. The encoder parameters are trained by back-propagating the critic loss into the encoder. The actor, critic, and encoder losses are minimized via gradient descent with mini-batches sampled from separate replay buffers for each task.

**Offline reinforcement learning.** In offline reinforcement learning, we assume that we have access to a dataset $\mathcal{D}$ collected by some policy behavior $\pi_\beta$. An RL agent must train on this fixed dataset and cannot interact with the environment. One challenge that offline RL poses is that the distribution of states and actions that an agent will see when deployed will likely be different from those seen in the offline dataset as they are generated by the agent, and a number of recent methods have tackled this distribution shift issue [73, 72, 121, 221, 157, 135]. Moreover, one can combine offline RL with meta-RL by training meta-RL on multiple datasets $\mathcal{D}_1, \ldots, \mathcal{D}_{N_{\text{buff}}}$ [50, 148], but in the next section we describe some limitations of this combination.

### 5.2.3 The Problem with Naïve Offline Meta-Reinforcement Learning

Although offline meta-RL methods must address the usual offline RL distribution shift issues, they must also contend with a distribution shift that is specific to the meta-RL scenario: distribution shift in $\mathbf{z}$-space. Distribution shift in $\mathbf{z}$-space occurs because meta-learning requires learning an exploration policy $\pi$ that generates data for adaptation. However, offline meta-learning only trains the adaptation procedure $A_\phi(\mathbf{h})$ using offline data $\mathbf{h}$ generated by a previous behavior policy, which we denote as $\pi_\beta$. After offline training, there will be a mismatch between this learned exploration policy $\pi$ and the behavior policy $\pi_\beta$, leading to a difference in the data $\mathbf{h}$ and in turn, in the context variables $\mathbf{z} = A_\phi(\mathbf{h})$. In other words, if we write $p(\mathbf{z} \mid \pi)$ to denote the marginal distribution over $\mathbf{z}$ given data generated by policy $\pi$, the differences between trajectories from $\pi$ and $\pi_\beta$ will result in differences between $p(\mathbf{z} \mid \pi_\beta)$ during offline training and $p(\mathbf{z} \mid \pi)$ at meta-test time.

To illustrate the presence of this distribution shift at meta-test time, we empirically compare $p(\mathbf{z} \mid \pi_\beta)$ and $p(\mathbf{z} \mid \pi)$. While computing these distributions in closed form would require strong assumptions about how the policy, adaptation procedure, and environment interact, we approximate the these distributions by using a PEARL-style encoder discussed in Section 5.2.2: $p(\mathbf{z} \mid \pi) \approx q_\phi(\mathbf{z} \mid \mathbf{h})$ where $\mathbf{h} \sim \pi$. We use this approximation to measure the KL-divergence observed during offline training between the posterior $p(\mathbf{z} \mid \pi)$ and a fixed prior $p(\mathbf{z})$. If these two distributions were the same, then we would expect the distribution of KL divergences to also be similar. However, we see in Figure 5.7 that these two distributions are markedly different when analyzing a training run of SMAC on the Ant Direction task (see Section 5.2.5 for details).

We also observe that this distribution shift negatively impacts the resulting policy. In Figure 5.7, we plot the performance of the learned policy when conditioned on $\mathbf{z}$ sampled from $q_\phi(\mathbf{z} \mid \pi_\beta)$ compared to $q_\phi(\mathbf{z} \mid \pi)$. We see that the policy conditioned on $\mathbf{z}$ generated from $\pi_\beta$ data leads to improvement, while the same policy conditioned on $\mathbf{z}$ generated from the exploration policy $\pi$ slightly drops in performance. Since we evaluate the same policy $\pi$ and only change how $\mathbf{z}$ is sampled, this degradation in performance suggests that the policy suffers from distributional shift between $p(\mathbf{z} \mid \pi_\beta)$ and $p(\mathbf{z} \mid \pi)$: the encoder produces $\mathbf{z}$

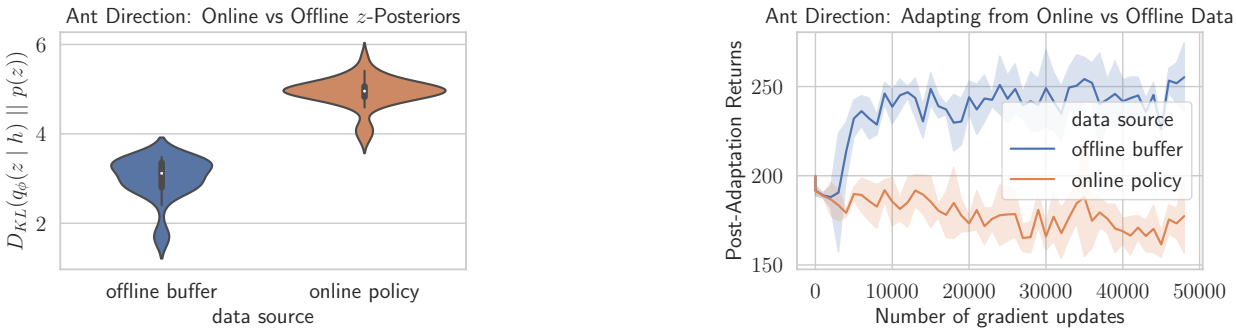



**Figure 5.7: Left:** The distribution of the KL-divergence between the posterior $q_\phi(\mathbf{z} \mid \mathbf{h})$ and a prior $p(\mathbf{z})$ over the course of training, when conditioned on data from the offline dataset (blue) or learned policy (orange), as measured by $D_{\text{KL}}(q_\phi(\mathbf{z} \mid \mathbf{h})||p(\mathbf{z}))$. We see that data from the learned policy results in posteriors that are substantially farther from the prior, suggesting a significant difference in distribution over $\mathbf{z}$. **Right:** The performance of the policy post-adaptation when conditioned on data from the offline dataset (i.e., $\mathbf{z} \sim p(\mathbf{z} \mid \pi_\beta)$) and data generated by the learned policy (i.e., $\mathbf{z} \sim p(\mathbf{z} \mid \pi)$). During the offline training phase, we see that although the meta-RL policy learns when conditioned on $\mathbf{z}$ generated by the offline data, the performance does not increase when $\mathbf{z}$ is generated using the online data. Since the same policy is evaluated, the change in $\mathbf{z}$-distribution is likely the cause for the drop in performance. In contrast, during the self-supervised training phase, the performance of the two is quite similar.

vectors that too unfamiliar to the policy after reading in these exploration trajectories, and therefore actually attains better performance when conditioned on the trajectories from $\pi_\beta$.

We note that this issue arises in any method for training non-Markovian policies with offline data. For example, recurrent policies for partially observed MDPs [100] depend both on the current observation $\mathbf{o}$ and a history $\mathbf{h}$. When deployed, these policies must also contend with potential distributional shifts between the training and test-time history distributions, in addition to the change in observation distribution $\mathbf{o}$. This additional distribution shift may explain why many memory-based recurrent policies are often trained online [52, 92, 57] or have benefited from refreshing the memory states [110]. In this paper, we focus on addressing this issue specifically in the offline meta-RL setting.

**Offline meta-RL with self-supervised online training.** In complex environments where many behaviors are possible, the distribution shift in $z$-space will likely be inevitable, since the learned policy is likely to deviate from the behavior policy. To address this issue, we introduce an additional assumption: in addition to the offline dataset, we assume that the agent can autonomously interact with the environment *without observing additional reward supervision*. This problem statement is useful for scenarios where autonomously interacting with the world is relatively easy, but online reward supervision is more expensive to obtain. For instance, it may be cheap to label rewards in an offline dataset for robotics by reward sketching [25], but expensive to have a labeler available online while the robot runs to provide rewards.

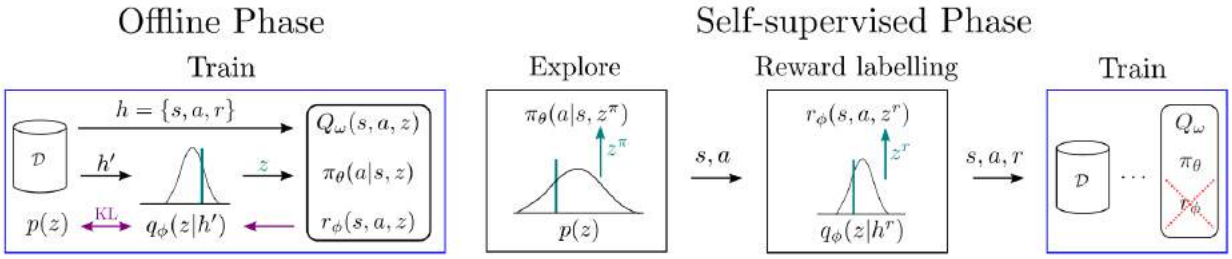



**Figure 5.8:** (Left) In the offline phase, we sample a history $\mathbf{h}'$ to compute the posterior $q_\phi(\mathbf{z} \mid \mathbf{h}')$. We then use a sample from this encoder and another history $\mathbf{h}$ to train the networks. In purple, we update the encoder $q_\phi$ with both reconstruction and KL loss. (Right) During the self-supervised phase, we explore by sampling $\mathbf{z} \sim p(z)$ and conditioning our policy on these observations. We label rewards using our learned reward decoder, and append the resulting data to the training data. The training procedure is equivalent to the offline phase, except that we do not train the reward decoder or encoder since no additional ground-truth rewards are observed.

Formally, we assume that the agent can generate additional rollouts in an MDP without a reward function, $\mathcal{T} \setminus r = (\mathcal{S}, \mathcal{A}, \gamma, p_0, p_d)$. These additional interactions enable the agent to explore using the learned policy. These exploration trajectories are from the same distribution that will be observed at meta-test time, and therefore can be included into the meta-training process to mitigate the distributional shift issue described above. However, meta-training requires not just states and actions, but also rewards. In the next section, we describe a method for autonomously labeling these rollouts with *synthetic* reward labels to enable an agent to meta-train on this additional data.

### 5.2.4 Semi-Supervised Meta Actor-Critic

In this section, we present semi-supervised meta actor-critic (SMAC), a method to perform offline meta-training followed by self-supervised online meta-training. For the offline meta-training, we assume access to a set of replay buffers, $\mathcal{D} = \{\mathcal{D}_i\}_{i=1}^{N_{\text{buff}}}$, where each buffer corresponds to data for one task. For self-supervised online meta-training, we assume that we can sample MDPs without a reward function. The SMAC adaptation procedure consists of passing history through the encoder described in Section 5.2.2, resulting in a posterior $q(\mathbf{z} \mid \mathbf{h})$. SMAC then uses this posterior for both meta-RL training and for reward generation. Below, we describe both components of the algorithm.

#### 5.2.4.1 Offline Meta-Training

To learn from the user-provided offline data, we adapt the PEARL meta-learning method [176] to the offline setting. We use an actor-critic algorithm to train a contextual policy using a set-based encoder, and update the critic by minimizing the Bellman error:

$$\mathcal{L}_{\text{critic}}(w) = \mathbb{E}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim\mathcal{D}_i, z\sim q_\phi(\mathbf{z}|\mathbf{h}), \mathbf{a}'\sim\pi(\mathbf{a}'|\mathbf{s}',\mathbf{z})} \left[ (Q_w(\mathbf{s},\mathbf{a},\mathbf{z}) - (r + \gamma Q_{\bar{w}}(\mathbf{s}',\mathbf{a}',\mathbf{z})))^2 \right], \quad (5.7)$$

where $\bar{w}$ are target network weights [149] updated with the soft update [138] of $\bar{w} \leftarrow \eta \cdot \bar{w} + (1 - \eta) \cdot w$.

PEARL uses soft actor critic (SAC) [85] to train their policy and $Q$-function. SAC has been primarily applied in the online setting, in which a replay buffer is continuously expanded by adding data from the latest policy. However, when naïvely applied to the offline setting, actor-critic methods such as SAC suffer from off-policy bootstrapping error accumulation [73, 121, 221], which occurs when the target $Q$-function for bootstrapping $Q(\mathbf{s}', \mathbf{a}')$ is evaluated at actions $\mathbf{a}'$ outside of the training data.

To avoid this error accumulation during offline training, we update our actor with a loss that implicitly constrains the policy to stay close to the actions observed in the replay buffer, following the approach in a previously proposed single-task offline RL algorithm called AWAC [157]. AWAC uses the following loss to approximate a constrained optimization problem, where the policy is constrained to stay close to the data observed in $\mathcal{D}$:

$$\mathcal{L}_{\text{actor}}(\theta) = \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}' \sim \mathcal{D}, \mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{h})} \left[ \log \pi(\mathbf{a} \mid \mathbf{s}) \exp\left( \frac{Q(\mathbf{s}, \mathbf{a}, \mathbf{z}) - V(\mathbf{s}', \mathbf{z})}{\lambda} \right) \right]. \tag{5.8}$$

We estimate the value function $V(\mathbf{s}, \mathbf{z}) = \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s},\mathbf{z})} Q(\mathbf{s}, \mathbf{a}, \mathbf{z})$ with a single sample, and $\lambda$ is the corresponding Lagrange multiplier for the optimization problem. See Nair et al. [157] for a full derivation.

This modified actor update makes it possible to train the encoder, actor, and critic on the offline data without the overestimation issues that afflict conventional actor-critic algorithms [121]. However, it does not address the $\mathbf{z}$-space distributional shift issue discussed in Section 5.2.3, because the exploration policy learned via this offline procedure will still deviate significantly from the behavior policy $\pi_\beta$. As discussed previously, we will aim to address this issue by collecting additional online data *without reward labels* and learning to generate reward labels if self-supervised meta-training.

**Learning to generate rewards.** To continue meta-training online without provided rewards, we propose to use the offline dataset to learn a generative model over meta-training task reward functions that we can use to label the transitions collected online. Recall that during offline learning, we learn an encoder $q_\phi$ which maps experience $\mathbf{h}$ to a latent context $\mathbf{z}$ that encodes the task. In the same way that we train our policy $\pi(\mathbf{a} \mid \mathbf{s}, \mathbf{z})$ that conditionally decodes $\mathbf{z}$ into actions, as well as a $Q$-function $Q_w(\mathbf{s}, \mathbf{a}, \mathbf{z})$ that conditionally decodes $\mathbf{z}$ into Q-values, we additionally train a *reward decoder* $p_\theta(\mathbf{s}, \mathbf{a}, \mathbf{z})$ [3] that conditionally decodes $\mathbf{z}$ into rewards. We train the reward decoder $p_\theta$ to reconstruct the observed reward in the offline dataset through a mean squared error loss.

Because we use the latent space $\mathbf{z}$ for reward-decoding, we back-propagate the reward decoder loss into $q_\phi$. As visualized in Figure 5.8, we also regularize the posteriors $q_\phi(\mathbf{z} \mid \mathbf{h})$ against a prior $p(\mathbf{z})$ to provide an information bottleneck in that latent space $\mathbf{z}$ and ensure that samples from $p(\mathbf{z})$ represent meaningful latent variables. We found it beneficial to not

[3] For simplicity, we write $\phi$ to represent the parameters of both the encoder and decoder.

back-propagate the critic loss into the encoder, in contrast to prior work such as PEARL. To summarize, we train the reward encoder and decoder by minimizing the following loss

$$\mathcal{L}_{\text{reward}}(\phi, \mathbf{h}, \mathbf{z}) = - \sum_{(\mathbf{s},\mathbf{a},r)\in\mathbf{h}} \|r - p_\theta(\mathbf{s}, \mathbf{a}, \mathbf{z})\|_2^2 + D_{\text{KL}}\left(q_\phi(\mathbf{z} \mid \mathbf{h}) \middle\| p(\mathbf{z})\right). \tag{5.9}$$

In the next section, describe how we use this reward decoder to generate new reward labels.

#### 5.2.4.2 Self-Supervised Meta-Training

We now describe the self-supervised online training procedure, during which we use the reward decoder to provide supervision. First, we collect a trajectory $\boldsymbol{\tau}$ by rolling out our exploration policy $\pi_\theta$ conditioned on a context sampled from the prior $p(\mathbf{z})$. To emulate the offline meta-training supervision, we would like to label $\boldsymbol{\tau}$ with rewards that are in the distribution of meta-training tasks. As such, we sample a replay buffer $\mathcal{D}_i$ uniformly from $\mathcal{D}$ to get a history $\mathbf{h} \sim \mathcal{D}_i$ from the offline data. We then sample from the posterior $\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{h})$ and label the reward $r_{\text{generated}}$ of a new state and action, $(\mathbf{s}, \mathbf{a})$, using the reward decoder

$$r_{\text{generated}} = p_\theta(\mathbf{s}, \mathbf{a}, \mathbf{z}), \qquad \text{where } \mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{h}) \tag{5.10}$$

We then add the labeled trajectory to the buffer and perform actor and critic updates as in offline meta-training. Lastly, since we do not observe additional ground-truth rewards, we do not update the reward decoder $p_\theta$ or encoder $q_\phi$, and instead only train the policy and $Q$-function during the self-supervised phase. We visualize this procedure in Figure 5.8.

**Algorithm 7** Semi-Supervised Meta Actor-Critic

Input: datasets $\mathcal{D} = \{\mathcal{D}_i\}_{i=1}^{N_{\text{buff}}}$, policy $\pi$, $Q$-function $Q_w$, encoder $q_\phi$, and decoder $p_\theta$.
**for** iteration $n = 1, 2, \ldots, N_{\text{offline}}$ **do**
  Sample buffer $\mathcal{D}_i \sim \mathcal{D}$ and two histories from buffer $\mathbf{h}, \mathbf{h}' \sim \mathcal{D}_i$.
  Use the first history sample to $\mathbf{h}$ to infer $\mathbf{z}$ encode it $\mathbf{z} \sim q_\phi(\mathbf{h})$.
  Update $\pi$, $Q_w$, $q_\phi$, $p_\theta$ by minimizing $\mathcal{L}_{\text{actor}}$, $\mathcal{L}_{\text{critic}}$, $\mathcal{L}_{\text{reward}}$ with samples $\mathbf{z}, \mathbf{h}'$.
**end for**
**for** iteration $n = 1, 2, \ldots, N_{\text{online}}$ **do**
  Collect trajectory $\boldsymbol{\tau}$ with $\pi(\mathbf{a} \mid \mathbf{s}, \mathbf{z})$, with $\mathbf{z}_t \sim p(\mathbf{z})$.
  Label the rewards in $\boldsymbol{\tau}$ using Equation (5.10) and add the resulting data to $\mathcal{D}_i$.
  Sample buffer $\mathcal{D}_i \sim \mathcal{D}$ and two histories from buffer $\mathbf{h}, \mathbf{h}' \sim \mathcal{D}_i$.
  Encode first history $\mathbf{z} = q_\phi(\mathbf{h})$.
  Update $\pi$, $Q_w$ by minimizing $\mathcal{L}_{\text{actor}}$, $\mathcal{L}_{\text{critic}}$ with samples $\mathbf{z}, \mathbf{h}'$.
**end for**

#### 5.2.4.3 Algorithm Summary and Details

We call the overall algorithm semi-supervised meta-actor-crici (SMAC) and visualize it in Figure 5.8. For offline training, we assume access to offline datasets $\mathcal{D} = \{\mathcal{D}_i\}_{i=1}^{N_{\text{buff}}}$, where each buffer corresponds to data generated for one task. Each iteration, we sample a buffer $\mathcal{D}_i \sim \mathcal{D}$ and a history from this buffer $\mathbf{h} \sim \mathcal{D}_i$. We condition the stochastic encoder $q_\phi$ on this history to obtain a sample $\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{h})$. We then use this sample $\mathbf{z}$ and a second history sample $\mathbf{h} \sim \mathcal{D}_i$ to update the $Q$-function, the policy, encoder, and decoder by minimizing Equation (5.7), Equation (5.8), and Equation (5.9) respectively. During the self-supervised phase, we found it beneficial to train the actor with a combination of the loss in Equation (5.8) and the original PEARL actor loss, weighted by hyperparameter $\lambda_{\text{pearl}}$. We provide pseudo-code for the method in Algorithm 7 a complete list of hyperparameters for reproducibility, such as the network architecture and RL hyperparameters, is provided in Appendix E.2.2.

### 5.2.5 Experiments

We proposed a method that uses additional online data to mitigate the distribution shift in $\mathbf{z}$-space that occurs in offline meta-RL. In this section, we evaluate whether or not the self-supervised phase of SMAC mitigates this negative drop in performance. We also study if SMAC can not only overcome the distribution shift problem, but also improve the ability to generalize to new tasks. In these experiments, we evaluate the adaptation procedure's ability to generalize to new task by testing it on held-out reward functions. We compare to different methods across multiple simulated robot domains.

**Meta-RL Tasks** We evaluate our method on multiple simulated MuJoCo [206] meta-learning tasks that have been used in past online and offline meta-RL papers [65, 176, 50, 148]. The first task, Cheetah Velocity, contains a two-legged "half cheetah" that can move forwards or backwards along the x-axis. Following prior work [176, 50], the reward function is the absolute difference product between the agent's x-velocity and a velocity uniformly sampled from $[0, 3]$. The second task, Ant Direction, contains a quadruped "ant" robot that can move in a plane. The reward function is the dot product between the agent's velocity and a direction uniformly sampled from the unit circle. In both of these domains, a meta-episode consists of sampling a desired velocity. The agent must learn to discover which velocity will maximize rewards within $T_{\text{adapt}} = 3$ episodes, each of length 200.

We also evaluated SMAC on a significantly more diverse robot manipulation meta-learning task called Sawyer Manipulation, based on the goal-conditioned environment introduced in Khazatsky et al. [114]. Sawyer Manipulation is a simulated PyBullet environment [38] which comprises a Sawyer robot arm that can manipulate drawers, pick and place objects, and push buttons. Sampling a task $\mathcal{T} \sim p(\mathcal{T})$ involves sampling both a new configuration of the environment and the desired behavior to achieve. The initial configuration of the objects can vary drastically, with the presence and location of objects randomized as shown in Figure 5.9 and the agent is tested on one of many possible desired behaviors, such as

**Figure 5.9:** Examples of our evaluation domains, each of which has a set of meta-train tasks (examples shown in blue) and held out test tasks (orange). The domains include (left) a half cheetah tasked with running at different speeds, (middle) a quadruped ant locomoting to different points on a circle, and (right) a simulated Sawyer arm performing various manipulation tasks.

pushing a button, opening a drawer, or lifting an object. The observation is a 13-dimensional state vector; when an object is not present in the task, the corresponding dimension takes on value 0. The action space is 4-dimensional: 3 dimensions to control the end-effector in Euclidean space and one dimension to control the gripper. The sparse reward is $-1$ when the desired behavior is not achieved and 0 when achieved. The task is difficult due to the diversity of objects, sparse reward, and precise manipulation required.

On all of the environments, we test the meta-RL procedure's ability to generalize to new tasks by evaluating the policies on *held-out tasks* sampled from the same distribution as in the offline datasets. We give a complete description of the possible tasks in Appendix E.2.2.

**Offline data collection.** For the MuJoCo tasks, we generate data by following a similar procedure as [73], in which we use the replay buffer from a single PEARL run that uses the ground-truth reward. We limit the data collection to 1200 transitions or 6 trajectories per task and terminate the PEARL run early, forcing the meta-RL agent to learn from sub-optimal data. For Sawyer Manipulation, we collect data using a scripted policy that randomly performs as many potential tasks in the environment, without knowing what the desired behavior in a sampled task is. We used 50 training tasks and 50 trajectories of length 75 per task. The average reward is -0.54 in the offline data. See Appendix E.2.2 for more details.

**Comparisons and ablations.** As an upper bound, we include the performance of PEARL when online training uses oracle ground-truth rewards rather than self-generated rewards, which we label *Online Oracle*. To understand the impact of using the actor loss in Equation (5.8), we include an ablation in which we use the actor loss from PEARL but still employ our proposed unsupervised online phase, which we label *SMAC (actor ablation)*. We also include a meta-imitation baseline, which infers the task like PEARL, but then imitates the task data in the dataset. In this baseline, we replace the actor update in Equation (5.8) with simply maximizing $\log \pi(\mathbf{a} \mid \mathbf{s}, \mathbf{z})$. We label this baseline *meta behavior cloning*. This baseline

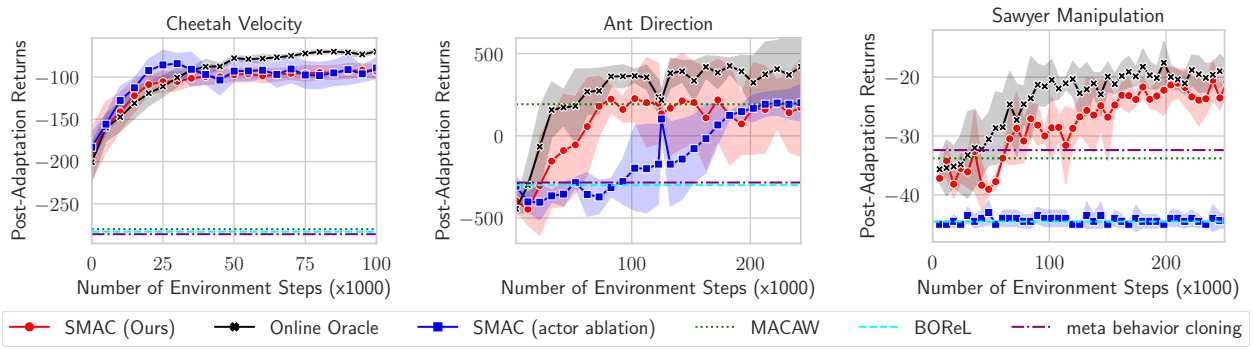



**Figure 5.10:** Comparison on self-supervised meta-learning against baseline methods. We report the final return of meta-test adaptation on unseen test tasks, with varying amounts of online meta-training following offline meta-training. Our method SMAC, shown in red, consistently trains to a reasonable performance from offline meta-RL (shown at step 0) and then steadily improves with online self-supervised experience. The offline meta-RL methods, MACAW [148] and BOReL at best match the offline performance of SMAC but have no mechanism to improve via self-supervision. We also compare to SMAC (SAC ablation) which uses SAC instead of AWAC as the underlying RL algorithm. This ablation struggles to pretrain a value function offline, and so struggles to improve on more difficult tasks.

illustrates the gap between offline meta-RL and imitation, and helps us understand the gap between the (highly suboptimal) behavior policy and RL.

For comparisons to prior work, we include the two previously proposed offline meta-RL methods: meta-actor critic with advantage weighting (labelled *MACAW*) [148] and Bayesian offline RL (labelled *BOReL*) [50]. Since these methods have only been applied to the offline phase, we report their performance only after offline training, since they do not have a self-supervised online stage. For both prior works, we used the code released by the authors. We trained these methods using the same offline dataset and matched hyperparameters when possible, such as batch size and network size.

**Comparison results.** We plot the mean post-adaptation returns and standard deviation across 4 seeds in Figure 5.10. We see that across all three environments, SMAC consistently improves during the self-supervised phase, and often achieves a similar performance to the oracle that uses ground-truth reward during the online phase of learning. SMAC also significantly improves over meta behavior cloning, which confirms that the data in the offline dataset is far from optimal.

We found that BOReL and MACAW performed comparatively poorly on all three tasks. A likely cause for this performance is that BOReL and MACAW were both developed assuming several orders of magnitude more data than the regime that we tested. For example, in the BOReL paper [50], the Cheetah Velocity was trained with an offline dataset using 400 million transitions and performs additional reward relabeling using ground-truth information about the transitions. In contrast, our offline dataset contains only 240 thousand transitions,

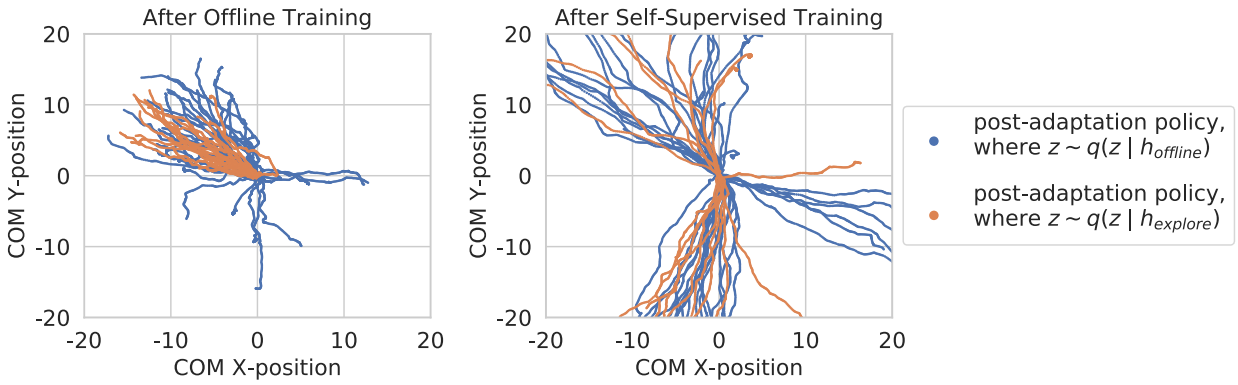



**Figure 5.11:** We visualize the visited XY-coordinates of the learned policy on the Ant Direction task. **Left:** Trajectories from the post-adaptation policy conditioned on $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{h})$ when $\mathbf{h}$ is sampled from the offline dataset (blue) or the learned exploration policy (orange) immediately after offline training. When conditioned on offline data, the policy correctly moves in many different directions. However, when conditioned on data from the learned exploration policy, the post-adaptation policy only moves up and to the left, suggesting that the post-adaptation policy is sensitive to data distribution used to collect $\mathbf{h}$. **Right:** After the self-supervised phase, we see that the post-adaptation policy learns to move in many different directions regardless of the data source. These visualization demonstrate that the self-supervised phase mitigates the distribution shift between conditioning on offline and online data.

roughly *three orders of magnitude* fewer transitions. Similarly, MACAW uses 100M transitions for Cheetah Velocity, over 40 times more transitions than used in our experiments. These prior methods also collect offline datasets by training *task-specific policies*, which converge to near-optimal policies within the first million time step [85], meaning that they are evaluated on very high-quality data.

In contrast, our data collection protocol produces more realistic offline datasets that are highly suboptimal, as evidenced by the performance of the meta behavior cloning baseline, leaving plenty of room for improvement with offline RL. We also observed that our method improves over the performance of BOReL and MACAW even *before* the online phase (i.e., at zero new environment steps) on the Cheetah and Sawyer Manipulation tasks, and achieves a particularly large improvement on the Sawyer Manipulation environments, which are by far the most challenging and exhibit the most variability between tasks. In this domain, we also see the largest gains from the AWAC actor update, in contrast to the actor ablation (in blue), indicating that properly handling the offline phase is also important for good performance.

**Visualizing the distribution shift.** Lastly, we investigate if the self-supervised training helps specifically because it mitigates a distribution shift caused by the exploration policy. To investigate this, we visualize the trajectories of the learned policy both before and after the self-supervised phase for the Ant Direction task in Figure 5.11. For each plot, we show trajectories from the policy $\pi(\mathbf{a} \mid \mathbf{s}, \mathbf{z})$ when the encoder $q_\phi(\mathbf{z} \mid \mathbf{h})$ is conditioned on histories

from either the offline dataset ($\mathbf{h}_{\text{offline}}$) or from the learned exploration policy ($\mathbf{h}_{\text{online}}$). Since the same policy is evaluated, differences between the resulting trajectories represent the distribution shift caused by using history from the learned exploration policy rather than from the offline dataset.

We see that before the self-supervised phase, there is a large difference between the two modes that can only be attributed to the difference in $\mathbf{h}$. When using $\mathbf{h}_{\text{online}}$, the post-adaptation policy only explores one mode, but when using $\mathbf{h}_{\text{offline}}$, the policy moves in all directions. This qualitative difference explains the large performance gap observed in Figure 5.7 and highlights that the adaptation procedure is sensitive to the history $\mathbf{h}$ used to adapt. In contrast, after the self-supervised phase, the policy moves in all directions regardless of where the history came from. In Appendix E.2.1, we also visualize the exploration trajectories and found that the exploration trajectories are qualitatively similar both before and after the self-supervised phase. Together, these results illustrate the SMAC policy learns to adapt to the exploration trajectories by using the self-supervised phase to mitigate the distribution shift that occurs with naïve offline meta RL.

## 5.3 Conclusion

We presented two example methods for training contextual policies beyond standard goal-reaching. We first presented DisCo RL, a method for learning distribution-conditioned, general-purpose policies specified using a goal distribution. An exciting direction for future work for DisCo would be to interleave DisCo RL and distribution learning, for example by using newly acquired data to update the learned VAE or by backpropagating the DisCo RL loss into the VAE learning loss. Another promising direction would be to study goal-distribution directed exploration, in which an agent can explore along a certain distribution of states, analogous to work on goal-directed exploration discussed in Section 3.2.

We also presented SMAC, a method for training meta-policies using offline data and autonomous environment interaction. We analyzed and addressed a specific problem in offline meta-RL: distribution shift in the context parameters $\mathbf{z}$. This distribution shift occurs because the data collected by a meta-learned exploration policy will differ from the data in an offline dataset. This difference is then magnified by the non-Markovian adaptation procedure in meta-RL, since the learned policy depends on the entire history through the post-adaptation context parameters $\mathbf{z}$. We provided evidence that this distribution shift both occurs and hurts the performance of offline meta-RL. To address this problem, we made an additional assumption that an agent can sample new trajectories without additional reward labels. We show experimentally that SMAC significantly improves over the performance of prior offline meta-RL methods, in many cases achieving better performance even after the offline phase alone, and significantly improving performance after the self-supervised online phase.

While our method significantly improves offline meta-RL performance, the most obvious limitation of SMAC is of course the need to gather additional unlabeled online samples.

This may be quite practical in domains where collecting reward labels is a major bottleneck, such as when rewards must be labeled by a human user, but may still pose a challenge in safety-critical settings where online interaction is impractical.

In addition to being useful for learning general-purpose policies and meta-policies, the research in Chapter 5 results highlight the challenges with and promising techniques for training contextual policies. For example, the work in Section 5.1 highlights the flexibility granted by learning conditional generative models over contexts. This insight suggests that conditional models could similarly grant flexibility in training goal-conditioned policies, as demonstrated in Nair et al. [156]. On the other hand, the work in Section 5.2 suggests that distribution shifts in the context space can drastically hurt the performance of contextual policies. This suggests a possible explanation for why techniques such as hindsight relabeling [104, 4] (see also Section 3.1) are so effective in goal-conditioned reinforcement: it is a form of data-augmentation in the context space that reduces the distribution shift between training and test time.

# Chapter 6

# Discussion and Future Work

In this thesis, we discussed how goal-conditioned reinforcement learning can be used to develop agents that can autonomously acquire reusable skills. We began in Chapter 2 by presenting the traditional goal-conditioned reinforcement learning problem and a novel method for training policies to reach a given goal. In Chapter 3, we then discussed how agents can autonomously generate goals for themselves that enable them to explore, especially in environments with high-dimensional states such as images. We continued in Chapter 4 by discussing how these goal-reaching policies can be reused in a planning framework. Specifically, we described how goal-conditioned policies provide an abstraction over how a goal is reached, enabling high-level planners to focus on what goals to reach using a temporal difference model. Lastly, we concluded in Chapter 5 by relating goal-conditioned reinforcement learning to more general problem formulations, such as maximizing arbitrary rewards or meta-learning.

**Future Work** Although Skew-Fit provides a principled approach for autonomous goal-setting in the absence of any prior knowledge, an important direction for future work is to incorporate *some* prior knowledge to guide exploration in real-world domains. Many real-world applications contain exponentially many states, rendering a uniform exploration strategy impractical. One possible remedy is to provide examples of interesting goal states before the agent autonomously explores. A goal-conditioned RL agent that has access to example goals can use these provided goals to guide its autonomous exploration to focus on regions of the state space that are of interest to the end user, similar to how SMAC uses a dataset with labeled rewards to generate similar rewards.

This relationship to SMAC and, more generally, Chapter 5 connects the techniques of goal-conditioned reinforcement learning to problem formulations involving contextual and Non-Markovian policies. We hope that highlighting this connection between contextual, meta, and goal-conditioned policies will encourage future cross-pollination of insights and techniques.

Combining goal-conditioned RL with planning is an exciting research direction that holds the potential to make RL methods more flexible, capable, and broadly applicable. Our work represents a step in this direction, though other crucial questions remain to be answered.

For example, how can planning be used to optimize the exploration objectives discussed in Section 3.2? How can we use more powerful constrained optimization methods for planning? Yet another promising research question is to study how lossy state abstractions might further improve the performance of the planner, by explicitly discarding state information that is irrelevant for higher-level planning. Exploring these future next steps can enable agents to solve complex tasks that may require more directed exploration, involved planning, or computationally efficient planning.

We also note that this work uses deep generative models for sampling and representing goals. One of the strengths of this approach is that many of the advances in generative model [88, 16] can be quickly translated to advances in goal-conditioned reinforcement learning. Exploring how more advanced generative models can be used to generate and represent goals in more complex domains or different modalities is a very promising next step, as evidence by the recent work that has explored this line of research [143, 123, 156, 170, 36, 113].

Finally, the work in this thesis presents a step in mitigating challenges associated with applying RL to many real-world systems. By enabling agents to autonomously acquire goal-directed behaviors, this work greatly minimizes the cost needed to train agents to acquire reusable skills. An exciting direction for future work is to apply these techniques to robotics applications "in the wild" and other control domains in which an agent must exhibit general-purpose capabilities.

# Bibliography


[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

[2] Arpit Agarwal, Katharina Muelling, and Katerina Fragkiadaki. Model learning for look-ahead exploration in continuous control. *AAAI*, 2019.

[3] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob Mcgrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems (NIPS)*, 2017. URL https://arxiv.org/pdf/1707.01495.pdfhttp://arxiv.org/abs/1707.01495.

[4] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob Mcgrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay. *arXiv preprint arXiv:1707.01495*, jul 2017. URL https://arxiv.org/pdf/1707.01495.pdfhttp://arxiv.org/abs/1707.01495.

[5] Karl Johan Åström and Richard M Murray. *Feedback systems*. Princeton university press, 2010.

[6] H. Attias. Planning by probabilistic inference. In *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*, 2003.

[7] Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H Campbell, and Sergey Levine. Stochastic variational video prediction. In *International Conference on Learning Representations*, 2018.

[8] Adrien Baranes and Pierre-Yves Oudeyer. Active Learning of Inverse Models with Intrinsically Motivated Goal Exploration in Robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2012. doi: 10.1016/j.robot.2012.05.008. URL http://dx.doi.org/10.1016/j.robot.2012.05.008.

[9] David Barber and Felix V Agakov. Information maximization in noisy channels: A variational approach. In *Advances in Neural Information Processing Systems*, 2004.

[10] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, pages 4055–4065, 2017.

[11] Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *ICML*, pages 501–510, 2018.

[12] André Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Žídek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. *arXiv preprint arXiv:1901.10964*, 2019.

[13] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1471–1479, 2016.

[14] Marc G Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C Machado, Subhodeep Moitra, Sameera S Ponda, and Ziyu Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, 2020.

[15] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[16] Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G Willcocks. Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models. *arXiv preprint arXiv:2103.04922*, 2021.

[17] Byron Boots, Arunkumar Byravan, and Dieter Fox. Learning predictive models of a depth camera & manipulator from raw execution traces. In *IEEE International Conference on Robotics and Automation*, pages 4021–4028, 2014. doi: 10.1109/ICRA.2014.6907443. URL http://dx.doi.org/10.1109/ICRA.2014.6907443.

[18] Diana Borsa, Andre Barreto, John Quan, Daniel J Mankowitz, Hado van Hasselt, Remi Munos, David Silver, and Tom Schaul. Universal successor features approximators. In *ICLR*, 2018.

[19] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[20] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[21] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[22] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

[23] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[24] Arunkumar Byravan, Felix Leeb, Franziska Meier, and Dieter Fox. SE3-Pose-Nets: Structured Deep Dynamics Models for Visuomotor Planning and Control. In *ICRA*, 2018. URL https://arxiv.org/pdf/1710.00489.pdf.

[25] Serkan Cabi, Sergio Gómez Colmenarejo, Alexander Novikov, Ksenia Konyushkova, Scott Reed, Rae Jeong, Konrad Zolna, Yusuf Aytar, David Budden, Mel Vecerik, et al. Scaling data-driven robotics with reward sketching and batch reinforcement learning. *arXiv preprint arXiv:1909.12200*, 2019.

[26] Yevgen Chebotar, Mrinal Kalakrishnan, Ali Yahya, Adrian Li, Stefan Schaal, and Sergey Levine. Path integral guided policy search. In *2017 IEEE International Conference on Robotics and Automation*. IEEE, 2017.

[27] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2172–2180, 2016. URL http://arxiv.org/abs/1606.03657.

[28] Yongxin Chen, Tryphon T. Georgiou, and Michele Pavon. Optimal steering of a linear stochastic system to a final probability distribution—part iii. *IEEE Transactions on Automatic Control*, 63(9):3112–3118, 2018. doi: 10.1109/TAC.2018.2791362.

[29] Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems*, 2005.

[30] Brian Cheung, Jesse A Livezey, Arjun K Bansal, and Bruno A Olshausen. Discovering hidden factors of variation in deep networks. *arXiv:1412.6583*, 2014.

[31] Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. In *International Conference on Learning Representations*, 2017.

[32] Jongwook Choi, Archit Sharma, Honglak Lee, Sergey Levine, and Shixiang Shane Gu. Variational Empowerment as Representation Learning for Goal-Based Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, 2021. URL https://arxiv.org/abs/2106.01404.

[33] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, 2018. URL https://arxiv.org/pdf/1805.12114.pdf.

[34] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms. *International Conference on Machine Learning (ICML)*, 2018.

[35] Cédric Colas, Pierre Fournier, Olivier Sigaud, and Pierre-Yves Oudeyer. CURIOUS: intrinsically motivated multi-task, multi-goal reinforcement learning. *International Conference on Machine Learning*, 2019.

[36] Cédric Colas, Ahmed Akakzia, Pierre-Yves Oudeyer, Mohamed Chetouani, and Olivier Sigaud. Language-conditioned goal generation: a new approach to language grounding for rl. *arXiv preprint arXiv:2006.07043*, 2020.

[37] Cédric Colas, Tristan Karch, Nicolas Lair, Jean-Michel Dussoux, Clément Moulin-Frier, Peter Ford Dominey, and Pierre-Yves Oudeyer. Language as a cognitive tool to imagine goals in curiosity-driven exploration. *arXiv preprint arXiv:2002.09253*, 2020.

[38] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

[39] Erwin Coumans et al. Bullet physics library. *Open source: bulletphysics. org*, 15(49):5, 2013.

[40] Bin Dai and David Wipf. Diagnosing and enhancing vae models. In *International Conference on Learning Representations*, 2019.

[41] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *NeurIPS*, pages 271–278, 1993.

[42] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1), 2005.

[43] Marc Peter Deisenroth and Carl Edward Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *ICML*, pages 465–472, 2011. URL http://mlg.eng.cam.ac.uk/pub/pdf/DeiRas11.pdf.

[44] Guillaume Desjardins, Aaron Courville, and Yoshua Bengio. Disentangling factors of variation via generative entangling. *CoRR*, abs/1210.5, 2012.

[45] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[46] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *arXiv preprint arXiv:2105.05233*, 2021.

[47] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13, 2000.

[48] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv:1605.08803*, 2016.

[49] Carl Doersch. Tutorial on variational autoencoders. *arXiv:1606.05908*, 2016.

[50] Ron Dorfman and Aviv Tamar. Offline meta reinforcement learning. *arXiv preprint arXiv:2008.02598*, 2020.

[51] Y. Duan, R. Chen, X. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

[52] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. $Rl^2$: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[53] Frederik Ebert, Chelsea Finn, Alex X Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. In *Conference on Robot Learning*, 2017. URL https://128.84.21.199/pdf/1710.05268.pdfhttps://arxiv.org/pdf/1710.05268.pdf.

[54] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018. URL https://sites.google.com/view/visualforesight.

[55] Ashley Edwards, Himanshu Sahni, Yannick Schroecker, and Charles Isbell. Imitating latent policies from observation. In *ICML*, pages 1755–1763, 2019.

[56] Scott Emmons, Ajay Jain, Michael Laskin, Thanard Kurutach, Pieter Abbeel, and Deepak Pathak. Sparse graphical memory for robust planning. *arXiv preprint arXiv:2003.06417*, 2020.

[57] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

[58] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is All You Need: Learning Skills without a Reward Function. In *International Conference on Learning Representations*, 2018. URL https://sites.google.com/view/diayn/.

[59] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. *arXiv preprint arXiv:1906.05253*, 2019. URL https://arxiv.org/pdf/1906.05253.pdf.

[60] Benjamin Eysenbach, Xinyang Geng, Sergey Levine, and Ruslan Salakhutdinov. Rewriting history with inverse rl: Hindsight inference for policy improvement. In *NeurIPS*, 2020.

[61] Matthew Fellows, Anuj Mahajan, Tim G. J. Rudner, and Shimon Whiteson. VIREL: A Variational Inference Framework for Reinforcement Learning. In *Advances in Neural Information Processing Systems 32*, 2019.

[62] Chelsea Finn and Sergey Levine. Deep Visual Foresight for Planning Robot Motion. In *Advances in Neural Information Processing Systems (NIPS)*, 2016. URL https://arxiv.org/pdf/1610.00696.pdfhttp://arxiv.org/abs/1610.00696.

[63] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58. PMLR, 2016.

[64] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *ICRA*, volume 2016-June, pages 512–519. IEEE, 2016. ISBN 9781467380263. doi: 10.1109/ICRA.2016.7487173.

[65] 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.

[66] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv:1704.03012*, 2017.

[67] Carlos Florensa, Jonas Degrave, Nicolas Heess, Jost Tobias Springenberg, and Martin Riedmiller. Self-supervised Learning of Image Embedding for Continuous Control. *arXiv:1901.00943*, 2018.

[68] Justin Fu, John D Co-Reyes, and Sergey Levine. EX 2 : Exploration with Exemplar Models for Deep Reinforcement Learning. In *Neural Information Processing Systems (NIPS)*, 2017.

[69] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018.

[70] Justin Fu, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. Variational inverse control with events: A general framework for data-driven reward definition. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *NeurIPS*, pages 8538–8547, 2018.

[71] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. *ICML*, 2018.

[72] Scott Fujimoto, Edoardo Conti, Mohammad Ghavamzadeh, and Joelle Pineau. Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint arXiv:1910.01708*, 2019.

[73] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

[74] Dibya Ghosh, Abhishek Gupta, and Sergey Levine. Learning actionable representations with goal-conditioned policies. In *ICLR*, 2018.

[75] Karolos M. Grigoriadis and Robert E. Skelton. Minimum-energy covariance controllers. *Automatica*, 33(4):569–578, 1997. ISSN 0005-1098. doi: https://doi.org/10.1016/S0005-1098(96)00188-4.

[76] Christopher Grimm, Irina Higgins, Andre Barreto, Denis Teplyashin, Markus Wulfmeier, Tim Hertweck, Raia Hadsell, and Satinder Singh. Disentangled cumulants help successor representations transfer to new tasks. *arXiv preprint arXiv:1911.10866*, 2019.

[77] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous Deep Q-Learning with Model-based Acceleration. In *ICML*, pages 2829–2838, 2016. ISBN 3405062780. doi: 10.3390/robotics2030122. URL https://arxiv.org/pdf/1603.00748.pdfhttp://arxiv.org/abs/1603.00748.

[78] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3389–3396. IEEE, 2017. ISBN 0896-6273. doi: 10.1038/nature20101. URL https://arxiv.org/pdf/1610.00633.pdfhttp://arxiv.org/abs/1610.00633.

[79] Abhishek Gupta, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Unsupervised meta-learning for reinforcement learning. *arXiv preprint arXiv:1806.04640*, 2018.

[80] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018.

[81] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Information Processing Systems*, pages 5302–5311, 2018. URL https://arxiv.org/pdf/1802.07245.pdf.

[82] David Ha and Jürgen Schmidhuber. World Models. *arXiv:1803.10122*, 2018.

[83] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *ICML*, 2018. URL https://arxiv.org/pdf/1801.01290.pdf.

[84] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2018.

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

[86] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, 2018.

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

[88] GM Harshvardhan, Mahendra Kumar Gourisaria, Manjusha Pandey, and Siddharth Swarup Rautaray. A comprehensive survey and analysis of generative models in machine learning. *Computer Science Review*, 38:100285, 2020.

[89] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

[90] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an Embedding Space for Transferable Robot Skills. In *ICLR*, pages 1–16, 2018.

[91] Elad Hazan, Sham M. Kakade, Karan Singh, and Abby Van Soest. Provably efficient maximum entropy exploration. *International Conference on Machine Learning (ICML)*, 2019.

[92] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[93] David Held, Xinyang Geng, Carlos Florensa, and Pieter Abbeel. Automatic Goal Generation for Reinforcement Learning Agents. In *International Conference on Machine Learning (ICML)*, 2018. URL `https://arxiv.org/pdf/1705.06366.pdf`.

[94] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *International Conference on Learning Representations*, 2016.

[95] Irina Higgins, Arka Pal, Andrei A Rusu, Loic Matthey, Christopher P Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. *ICML*, 2017.

[96] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *arXiv preprint arXiv:1606.03476*, 2016.

[97] Matthew Hoffman, Nando Freitas, Arnaud Doucet, and Jan Peters. An expectation maximization algorithm for continuous markov decision processes with arbitrary reward. In *Artificial intelligence and statistics*, pages 232–239, 2009.

[98] Anthony F. Hotz and Robert E. Skelton. A covariance control theory. In C.T. LEONDES, editor, *System Identification and Adaptive Control, Part 2 of 3*, volume 26 of *Control and Dynamic Systems*, pages 225–276. Academic Press, 1987. doi: https://doi.org/10.1016/B978-0-12-012726-9.50010-8.

[99] Jan Humplik, Alexandre Galashov, Leonard Hasenclever, Pedro A Ortega, Yee Whye Teh, and Nicolas Heess. Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424*, 2019.

[100] Tommi Jaakkola, Satinder P Singh, and Michael I Jordan. Reinforcement learning algorithm for partially observable markov decision problems. *Advances in neural information processing systems*, pages 345–352, 1995.

[101] Allan Jabri, Kyle Hsu, Ben Eysenbach, Abhishek Gupta, Sergey Levine, and Chelsea Finn. Unsupervised curricula for visual meta-reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 10519–10531, 2019.

[102] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, 2019.

[103] Rico Jonschkowski, Roland Hafner, Jonathan Scholz, and Martin Riedmiller. Pves: Position-velocity encoders for unsupervised learning of structured state representations. *arXiv preprint arXiv:1705.09805*, 2017.

[104] L P Kaelbling. Learning to achieve goals. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume vol.2, pages 1094 – 8, 1993.

[105] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[106] Mrinal Kalakrishnan, Ludovic Righetti, Peter Pastor, and Stefan Schaal. Learning force control policies for compliant manipulation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011.

[107] Nal Kalchbrenner, Aäron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. In *International Conference on Machine Learning*, 2017.

[108] Pierre-Alexandre Kamienny, Matteo Pirotta, Alessandro Lazaric, Thibault Lavril, Nicolas Usunier, and Ludovic Denoyer. Learning adaptive exploration strategies in dynamic environments through informed policy regularization. *arXiv preprint arXiv:2005.02934*, 2020.

[109] H. J. Kappen, V. Gómez, and M. Opper. Optimal control as a graphical model inference problem. *Machine Learning*, 87(2):159–182, 2012.

[110] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

[111] Miroslav Kárnỳ. Towards fully probabilistic control design. *Automatica*, 32(12):1719–1722, 1996.

[112] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[113] Alexander Khazatsky, Ashvin Nair, Daniel Jing, and Sergey Levine. What can i do here? learning new skills by imagining visual affordances. *arXiv preprint arXiv:2106.00671*, 2021.

[114] Alexander Khazatsky, Ashvin Nair, Daniel Jing, and Sergey Levine. What can i do here? learning new skills by imagining visual affordances. In *International Conference on Robotics and Automation*. IEEE, 2021.

[115] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[116] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. In *ICLR*, 2014. URL https://arxiv.org/pdf/1312.6114.pdf.

[117] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *NeurIPS*, pages 10215–10224, 2018.

[118] Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *arXiv preprint arXiv:1910.04098*, 2019.

[119] Ksenia Konyushkova, Konrad Zolna, Yusuf Aytar, Alexander Novikov, Scott Reed, Serkan Cabi, and Nando de Freitas. Semi-supervised reward learning for offline reinforcement learning. *arXiv preprint arXiv:2012.06899*, 2020.

[120] Tejas D Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J Gershman. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.

[121] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *arXiv preprint arXiv:1906.00949*, 2019.

[122] Thanard Kurutach, Aviv Tamar, Ge Yang, Stuart J Russell, and Pieter Abbeel. Learning plannable representations with causal infogan. In *Advances in Neural Information Processing Systems*, pages 8733–8744, 2018.

[123] Nicolas Lair, Cédric Colas, Rémy Portelas, Jean-Michel Dussoux, Peter Ford Dominey, and Pierre-Yves Oudeyer. Language grounding through social interactions and curiosity-driven multi-goal learning. *arXiv preprint arXiv:1911.03219*, 2019.

[124] Terran Lane and Leslie Pack Kaelbling. Toward hierarchical decomposition for planning in uncertain environments. In *Proceedings of the 2001 IJCAI workshop on planning under uncertainty and incomplete information*, 2001.

[125] Sascha Lange and Martin A Riedmiller. Deep learning of visual control policies. In *European Symposium on Artificial Neural Networks (ESANN)*. Citeseer, 2010. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.226.6898{&}rep=rep1{&}type=pdf.

[126] Sascha Lange, Martin Riedmiller, Arne Voigtlander, and Arne Voigtländer. Autonomous reinforcement learning on raw visual input data in a real world application. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2012. ISBN 9781467314909. doi: 10.1109/IJCNN.2012.6252823.

[127] Nevena Lazic, Tyler Lu, Craig Boutilier, MK Ryu, Eehern Jay Wong, Binz Roy, and Greg Imwalle. Data center cooling using model-predictive control. In *Advances in Neural Information Processing Systems*, 2018.

[128] Alex Lee, Sergey Levine, and Pieter Abbeel. Learning Visual Servoing with Deep Features and Fitted Q-Iteration. In *ICLR*, 2017. URL https://arxiv.org/pdf/1703.11000.pdf.

[129] Alex X Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv:1907.00953*, 2019.

[130] Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching. In *ICLR*, 2019.

[131] Youngwoon Lee, Edward S Hu, Zhengyu Yang, Alex Yin, and Joseph J Lim. IKEA furniture assembly environment for long-horizon complex manipulation tasks. *arXiv:1911.07246*, 2019. URL https://clvrai.com/furniture.

[132] Ian Lenz, Ross Knepper, and Ashutosh Saxena. DeepMPC: Learning Deep Latent Features for Model Predictive Control. In *RSS*, 2015.

[133] Sergey Levine. Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review. *arXiv preprint arXiv:1805.00909*, 2018. URL https://arxiv.org/pdf/1805.00909.pdf.

[134] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-End Training of Deep Visuomotor Policies. *Journal of Machine Learning Research (JMLR)*, 17(1): 1334–1373, 2016. ISSN 15337928. URL https://arxiv.org/pdf/1504.00702.pdf.

[135] 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.

[136] Andrew Levy, Robert Platt, and Kate Saenko. Hierarchical Actor-Critic. *arXiv:1712.00948*, 2017.

[137] Yuanlong Li, Yonggang Wen, Dacheng Tao, and Kyle Guan. Transforming cooling optimization for green data center via deep reinforcement learning. *IEEE transactions on cybernetics*, 50(5):2002–2013, 2019.

[138] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2016. ISSN 10769757. doi: 10.1613/jair.301. URL https://arxiv.org/pdf/1509.02971.pdf.

[139] Kara Liu, Thanard Kurutach, Christine Tung, Pieter Abbeel, and Aviv Tamar. Hallucinative topological memory for zero-shot visual planning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*. PMLR, 2020.

[140] Yuxuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from Observation: Learning to Imitate Behaviors from Raw Video via Context Translation. In *ICRA*, 2018. URL https://arxiv.org/pdf/1707.03374.pdf.

[141] Manuel Lopes, Tobias Lang, Marc Toussaint, and Pierre-Yves Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Advances in Neural Information Processing Systems*, 2012.

[142] Haw-Minn Lu, Yeshaiahu Fainman, and Robert Hecht-Nielsen. Image manifolds. In *Applications of Artificial Neural Networks in Image Processing III*, volume 3307. International Society for Optics and Photonics, 1998.

[143] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. *Conference on Robot Learning (CoRL)*, 2019. URL https://arxiv.org/abs/1903.01973.

[144] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. In *International Conference on Learning Representations*, 2016.

[145] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. In *International Conference on Learning Representations*, 2016.

[146] Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*, 2017.

[147] Thomas P Minka. Expectation propagation for approximate bayesian inference. In *UAI*, pages 362–369, 2001.

[148] Eric Mitchell, Rafael Rafailov, Xue Bin Peng, Sergey Levine, and Chelsea Finn. Offline meta-reinforcement learning with advantage weighting. *arXiv preprint arXiv:2008.06043*, 2020.

[149] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and Others. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[150] Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 28, pages 2125–2133, 2015.

[151] Richard M Murray, Zexiang Li, and S Shankar Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[152] Ofir Nachum, Google Brain, Shane Gu, Honglak Lee, and Sergey Levine. Data-Efficient Hierarchical Reinforcement Learning. In *NeurIPS*, 2018. URL https://sites.google.com/view/efficient-hrl.

[153] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. *arXiv preprint arXiv:1708.02596*, 2018. URL https://arxiv.org/pdf/1708.02596.pdf.

[154] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *IEEE International Conference on Robotics and Automation*, 2018.

[155] Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual Reinforcement Learning with Imagined Goals. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[156] Ashvin Nair, Shikhar Bahl, Alexander Khazatsky, Vitchyr Pong, Glen Berseth, and Sergey Levine. Contextual imagined goals for self-supervised robotic learning. In *CoRL*, pages 530–539, 2020.

[157] Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[158] Soroush Nasiriany, Vitchyr H Pong, Ashvin Nair, Alexander Khazatsky, Glen Berseth, and Sergey Levine. Disco rl: Distribution-conditioned reinforcement learning for general-purpose policies. In *IEEE International Conference on Robotics and Automation*, 2014.

[159] Soroush Nasiriany, Vitchyr H Pong, Steven Lin, and Sergey Levine. Planning with goal-conditioned policies. In *Advances in Neural Information Processing Systems 29*, 2019.

[160] Frank Nielsen and Richard Nock. Entropies and cross-entropies of exponential families. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*. IEEE, 2010.

[161] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[162] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard Lewis, and Satinder Singh. Action-Conditional Video Prediction using Deep Networks in Atari Games. In *Advances in Neural Information Processing Systems (NIPS)*, 2015. URL https://arxiv.org/pdf/1507.08750v1.pdf.

[163] Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International Conference on Machine Learning*, 2017.

[164] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3), 2014.

[165] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[166] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-Driven Exploration by Self-Supervised Prediction. In *ICML*, pages 488–489. IEEE, 2017.

[167] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *RSS*, 2020.

[168] Alexandre Péré, Sebastien Forestier, Olivier Sigaud, and Pierre-Yves Oudeyer. Unsupervised Learning of Goal Spaces for Intrinsically Motivated Goal Exploration. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://arxiv.org/pdf/1803.00781.pdf.

[169] J. Peters, K. Mülling, and Y. Altün. Relative entropy policy search. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 1607–1612, 2010. URL https://pdfs.semanticscholar.org/ff47/526838ce85d77a50197a0c5f6ee5095156aa.pdfhttp://www-clmc.usc.edu/publications/P/Peters{_}POTTNCOAIPGAT{_}2010.pdf.

[170] Silviu Pitis, Harris Chan, Stephen Zhao, Bradly Stadie, and Jimmy Ba. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In *International Conference on Machine Learning*, pages 7750–7761. PMLR, 2020.

[171] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob Mcgrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research. *arXiv:1802.09464*, 2018. URL http://fetchrobotics.com/.

[172] Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal Difference Models: Model-Free Deep RL For Model-Based Control. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://arxiv.org/pdf/1802.09081.pdf.

[173] Vitchyr H. Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *CoRR*, abs/1903.03698, 2020. URL http://arxiv.org/pdf/1903.03698.pdf.

[174] Nikolay Ponomarenko, Lina Jin, Oleg Ieremeiev, Vladimir Lukin, Karen Egiazarian, Jaakko Astola, Benoit Vozel, Kacem Chehdi, Marco Carli, Federica Battisti, and Others. Image database TID2013: Peculiarities, results and perspectives. *Signal Processing: Image Communication*, 30:57–77, 2015.

[175] Ali Punjani and Pieter Abbeel. Deep learning helicopter dynamics models. In *IEEE International Conference on Robotics and Automation*, 2015.

[176] 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.

[177] Paulo Rauber, Filipe Mutz, and Juergen Jürgen Schmidhuber. Hindsight policy gradients. In *CoRR*, volume abs/1711.0, 2017. URL https://arxiv.org/pdf/1711.06006.pdf.

[178] K. Rawlik, M. Toussaint, and S. Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *Robotics: Science and Systems (RSS)*, 2013. URL http://roboticsproceedings.org/rss08/p45.pdf.

[179] Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. An approximate inference approach to temporal optimization in optimal control. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. URL https://proceedings.neurips.cc/paper/2010/file/b5dc4e5d9b495d0196f61d45b26ef33e-Paper.pdf.

[180] Siddharth Reddy, Anca D Dragan, and Sergey Levine. Sqil: Imitation learning via reinforcement learning with sparse rewards. *arXiv preprint arXiv:1905.11108*, 2019.

[181] Scott Reed, Kihyuk Sohn, Yuting Zhang, and Honglak Lee. Learning to disentangle factors of variation with manifold interaction. In *ICML*, pages 1431–1439, 2014.

[182] Danilo J Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *ICML*, 2014. URL https://arxiv.org/pdf/1401.4082.pdf.

[183] Jack Ridderhof, Kazuhide Okamoto, and Panagiotis Tsiotras. Nonlinear uncertainty control with iterative covariance steering, 2019.

[184] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR Workshop and Conference Proceedings, 2010.

[185] Donald B Rubin. Using the sir algorithm to simulate posterior distributions. *Bayesian statistics*, 3:395–402, 1988.

[186] Tim GJ Rudner, Vitchyr H Pong, Rowan McAllister, Yarin Gal, and Sergey Levine. Outcome-driven reinforcement learning via variational inference. *arXiv preprint arXiv:2104.10190*, 2021.

[187] Andrei A Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *CoRL*, 2017.

[188] Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274*, 2018.

[189] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

[190] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal Value Function Approximators. In *ICML*, pages 1312–1320, 2015. ISBN 9781510810587. URL http://proceedings.mlr.press/v37/schaul15.pdf.

[191] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal Value Function Approximators. In *International Conference on Machine Learning (ICML)*, 2015.

[192] Jürgen Schmidhuber. Learning factorial codes by predictability minimization. *Neural Computation*, 4(6):863–879, 1992.

[193] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[194] Yannick Schroecker and Charles Isbell. Universal value density estimation for imitation learning and goal-conditioned reinforcement learning. *arXiv preprint arXiv:2002.06473*, 2020.

[195] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. *arXiv preprint arXiv:1704.06888*, 2018.

[196] Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine. End-to-end robotic reinforcement learning without reward engineering. *arXiv preprint arXiv:1904.07854*, 2019. URL http://arxiv.org/abs/1904.07854.

[197] Linda Smith and Michael Gasser. The development of embodied cognition: Six lessons from babies. *Artificial life*, 11(1-2):13–29, 2005. URL https://www.cogsci.msu.edu/DSS/2010-2011/Smith/6lessons.pdf.

[198] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. *arXiv preprint arXiv:1804.00645*, 2018.

[199] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models. In *ICLR*, 2016. URL https://arxiv.org/pdf/1507.00814.pdf.

[200] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[201] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2), 1999.

[202] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 10, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011. URL https://www.cs.swarthmore.edu/{~}meeden/DevelopmentalRobotics/horde1.pdf.

[203] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. #Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning. In *Neural Information Processing Systems (NIPS)*, nov 2016. URL http://arxiv.org/abs/1611.04717.

[204] Valentin Thomas, Jules Pondard, Emmanuel Bengio, Marc Sarfati, Philippe Beaudoin, Marie-Jean Meurs, Joelle Pineau, Doina Precup, Yoshua Bengio, Valentin Thoma, Joelle Pineau, Doina Precup, and Yoshua Bengio. Independently Controllable Factors. In *NIPS Workshop*, 2017. URL https://arxiv.org/pdf/1703.07718.pdfhttps://arxiv.org/pdf/1708.01289.pdf.

[205] E. Todorov. Linearly-solvable Markov decision problems. In *Neural Information Processing Systems (NeurIPS)*, 2006. URL http://papers.nips.cc/paper/3002-linearly-solvable-markov-decision-problems.pdf.

[206] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. ISBN 9781467317375. doi: 10.1109/IROS.2012.6386109. URL https://homes.cs.washington.edu/{~}todorov/papers/TodorovIROS12.pdf.

[207] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In *IJCAI*, 2018.

[208] Faraz Torabi, Garrett Warnell, and Peter Stone. Generative adversarial imitation from observation. *arXiv:1807.06158*, 2018.

[209] M. Toussaint. Robot trajectory optimization using approximate inference. In *International Conference on Machine Learning (ICML)*, 2009. URL https://icml.cc/Conferences/2009/papers/271.pdf.

[210] Marc Toussaint and Amos Storkey. Probabilistic inference for solving discrete and continuous state markov decision processes. In *Proceedings of the 23rd international conference on Machine learning*, pages 945–952, 2006.

[211] Marc Toussaint, Stefan Harmeling, and Amos Storkey. Probabilistic inference for solving (po)mdps. Technical report, Technical Report EDI-INF-RR-0934, School of Informatics, University of Edinburgh, 2006.

[212] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *NeurIPS*, pages 4790–4798, 2016.

[213] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[214] Vivek Veeriah, Junhyuk Oh, and Satinder Singh. Many-goals reinforcement learning. *arXiv preprint arXiv:1806.09605*, 2018.

[215] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, 2017. URL https://arxiv.org/pdf/1703.01161.pdfhttp://arxiv.org/abs/1703.01161.

[216] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[217] David Warde-Farley, Tom Van De Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Mnih Volodymyr. Unsupervised Control Through Non-Parametric Discriminative Rewards. In *ICLR*, 2019.

[218] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2728–2736, 2015. URL https://arxiv.org/pdf/1506.07365.pdfhttp://arxiv.org/abs/1506.07365.

[219] Martha White. Unifying task specification in reinforcement learning. In *International Conference on Machine Learning*, pages 3742–3750. PMLR, 2017.

[220] Marco Wiering and Jürgen Schmidhuber. Hq-learning. *Adaptive Behavior*, 6(2), 1997.

[221] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[222] Danfei Xu and Misha Denil. Positive-unlabeled reward learning. *arXiv preprint arXiv:1911.00459*, 2019.

[223] J.-H. Xu and R.E. Skelton. An improved covariance assignment theory for discrete systems. *IEEE Transactions on Automatic Control*, 37(10):1588–1591, 1992. doi: 10.1109/9.256389.

[224] Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. *Advances in neural information processing systems*, 31:2396–2407, 2018.

[225] Zhongwen Xu, Hado van Hasselt, Matteo Hessel, Junhyuk Oh, Satinder Singh, and David Silver. Meta-gradient reinforcement learning with an objective discovered online. *arXiv preprint arXiv:2007.08433*, 2020.

[226] Z. Yi, Z. Cao, E. Theodorou, and Y. Chen. Nonlinear covariance control via differential dynamic programming. In *2020 American Control Conference (ACC)*, pages 3571–3576, 2020. doi: 10.23919/ACC45564.2020.9147531.

[227] 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.

[228] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond Empirical Risk Minimization. In *ICLR*, oct 2018. URL http://arxiv.org/abs/1710.09412.

[229] Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew J. Johnson, and Sergey Levine. SOLAR: Deep Structured Representations for Model-Based Reinforcement Learning. In *ICML*, 2019. URL http://arxiv.org/abs/1808.09105.

[230] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[231] Rui Zhao and Volker Tresp. Curiosity-driven experience prioritization via density estimation. *CoRR*, abs/1902.08039, 2019. URL http://arxiv.org/abs/1902.08039.

[232] Rui Zhao, Xudong Sun, and Volker Tresp. Maximum entropy-regularized multi-goal reinforcement learning. In *International Conference on Machine Learning*, pages 7553–7562, 2019.

[233] Tony Z Zhao, Anusha Nagabandi, Kate Rakelly, Chelsea Finn, and Sergey Levine. Meld: Meta-reinforcement learning from images via latent state models. *arXiv preprint arXiv:2010.13957*, 2020.

[234] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *International Conference on Artificial Intelligence (AAAI)*, pages 1433–1438, 2008. ISBN 9781577353683 (ISBN). URL https://www.aaai.org/Papers/AAAI/2008/AAAI08-227.pdfhttp://www.scopus.com/inward/record.url?eid=2-s2.0-57749097473{&}partnerID=40{%}5Cnhttp://www.aaai.org/Papers/AAAI/2008/AAAI08-227.pdf.

[235] Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: a very good method for bayes-adaptive deep rl via meta-learning. *Proceedings of ICLR 2020*, 2020.

[236] Konrad Zolna, Alexander Novikov, Ksenia Konyushkova, Caglar Gulcehre, Ziyu Wang, Yusuf Aytar, Misha Denil, Nando de Freitas, and Scott Reed. Offline learning from demonstrations and unlabeled experience. *arXiv preprint arXiv:2011.13885*, 2020.

# Appendix A

# Contributions

- Section 2.3 is based on Rudner et al. [186], co-first authored with Tim G. J. Rudner. Sergey Levine proposed the idea of treating goal-conditioning reinforcement learning as variational inference. Tim G. J. Rudner wrote the first version of the proofs, demonstrating goal-conditioned reinforcement learning objectives can be derived from variational inference over trajectory distributions. Vitchyr H. Pong proposed to treat the time at which the goal is reached as a separate random variable and to infer it variationally alongside the distribution over the goal-conditioned trajectory. Tim unified this approach with the previous results by proposing to treat the trajectory as a transdimensional random variable. Together, Vitchyr and Tim significantly revised and extended the proofs, with Tim proving the majority of the results. Tim and Vitchyr worked together on designing the experiments, with Vitchyr conducting the experiments. Sergey Levine, Rowan McAllister, and Yarin Gal provided feedback on the paper drafts and contributed to the model formulation and variational inference method.
- Section 3.1 is based on Nair et al. [155], co-first authored with Ashvin Nair.
- Section 3.2 is based on Pong et al. [173], co-first authored with Murtaza Dalal and Steven Lin.
- Section 4.1 is based on Pong et al. [172], co-first authored with Shixiang Gu.
- Section 4.2 is based on Nasiriany et al. [159], co-first authored with Soroush Nasiriany.
- Section 5.1 is based on Nasiriany et al. [158], co-first authored with Soroush Nasiriany and Ashvin Nair.

I especially thank these collaborators for their significant contributions to this thesis!

# Appendix B

# Chapter 2 Appendix

## B.1 Proofs & Derivations

### B.1.1 Derivation of Variational Objectives

In this section, we present detailed derivations to complement the derivations and results included in Section 2.3.2 and Section 2.3.3.

**Proposition 1** (Fixed-Time Outcome-Driven Variational Objective)**.** *Let* $q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0)$ *be as defined in Equation (2.7). Then, given any initial state* $\mathbf{s}_0$*, termination time* $t^\star$*, and outcome* $\mathbf{g}$*,*

$$D_{\mathrm{KL}}(q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t, \mathbf{s}_0) \,\|\, p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t, \mathbf{s}_0, \mathbf{s}_{t^\star} = \mathbf{g})) = \log p(\mathbf{g} \mid \mathbf{s}_0) - \bar{\mathcal{F}}(\pi, \mathbf{s}_0, \mathbf{g}), \tag{B.1}$$

*where*

$$\bar{\mathcal{F}}(\pi, \mathbf{s}_0, \mathbf{g}) \doteq \mathbb{E}_{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot|\mathbf{s}_0)}\left[\log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) + \sum_{t'=0}^{t-1} D_{\mathrm{KL}}(\pi(\cdot \mid \mathbf{s}_{t'}) \,||\, p(\cdot \mid \mathbf{s}_{t'}))\right], \tag{B.2}$$

*and since* $\log p(\mathbf{g}|\mathbf{s}_0)$ *is constant in* $\pi$*,*

$$\arg\min_{\pi \in \Pi} D_{\mathrm{KL}}(q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t, \mathbf{s}_0) \,\|\, p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t, \mathbf{s}_0, \mathbf{s}_{t^\star} = \mathbf{g})) = \arg\max_{\pi \in \Pi} \bar{\mathcal{F}}(\pi, \mathbf{s}_0, \mathbf{g}). \tag{B.3}$$

*Proof.* To find the (approximate) posterior $p_{\mathbf{a}_t}(\cdot \mid \mathbf{s}_t, \mathbf{s}_{t^\star} = \mathbf{g})$, we can use variational inference. To do so, we consider the trajectory distribution under $p_{\mathbf{a}_t}(\cdot \mid \mathbf{s}_t, \mathbf{s}_{t^\star} = \mathbf{g})$,

$$p(\tilde{\boldsymbol{\tau}}_{0:t} \mid \mathbf{s}_0, \mathbf{s}_{t^\star} = \mathbf{g}) = p(\mathbf{a}_t \mid \mathbf{s}_t, \mathbf{s}_{t^\star} = \mathbf{g}) \prod_{t'=0}^{t-1} p_d(\mathbf{s}_{t'+1} \mid \mathbf{s}_t, \mathbf{a}_t) p(\mathbf{a}_{t'} \mid \mathbf{s}_{t'}, \mathbf{s}_{t^\star} = \mathbf{g}), \tag{B.4}$$

where $t = t^\star - 1$ and we denote the state–action trajectory realization from state $\mathbf{s}_0$ and action $\mathbf{a}_0$ to time $t^\star - 1$ by $\tilde{\boldsymbol{\tau}}_{0:t} \doteq \{\mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, ..., \mathbf{s}_t, \mathbf{a}_t\}$. Inferring a posterior distribution

$p(\tilde{\boldsymbol{\tau}}_{0:t}|\mathbf{s}_{t^\star} = \mathbf{g})$ then becomes equivalent to finding a variational distribution $q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid \mathbf{s}_0)$, which induces a trajectory distribution $q(\tilde{\boldsymbol{\tau}}_0|\mathbf{s}_0)$ that minimizes the KL divergence from $q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid \mathbf{s}_0)$ to $p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t, \mathbf{s}_0, \mathbf{s}_{t^\star} = \mathbf{g})$:

$$\min_{q \in \bar{\mathcal{Q}}} D_{\mathrm{KL}} \left( q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t, \mathbf{s}_0) \middle\| p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t, \mathbf{s}_0, \mathbf{s}_{t^\star} = \mathbf{g}) \right). \tag{B.5}$$

If we find a distribution $q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid \mathbf{s}_0)$ for which the resulting KL divergence is zero, then $q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid \mathbf{s}_0)$ is the exact posterior. If the KL divergence is positive, then $q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid \mathbf{s}_0)$ is an approximate posterior. To solve the variational inference problem in Equation (B.5), we can define a factorized variational family as

$$q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid \mathbf{s}_0) \doteq \pi(\mathbf{a}_t \mid \mathbf{s}_t) \prod_{t'=0}^{t-1} q(\mathbf{s}_{t'+1} \mid \mathbf{s}_{t'}, \mathbf{a}_{t'}) \pi(\mathbf{a}_{t'} \mid \mathbf{s}_{t'}), \tag{B.6}$$

where $\mathbf{a}_{0:t}$ are latent variables to be inferred, and the product is from $t = 0$ to $t = t - 1$ to exclude the conditional distribution over the (observed) state $\mathbf{s}_{t+1} = \mathbf{g}$ from the variational distribution.

Returning to the variational problem in Equation (B.5), we can now write

$$\begin{aligned} D_{\mathrm{KL}} \left( q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid \mathbf{s}_0) \middle\| p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t, \mathbf{s}_0, \mathbf{s}_{t^\star} = \mathbf{g}) \right) &= \int_{\mathcal{A}^t} \int_{\mathcal{S}^t} q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid \mathbf{s}_0) \log \frac{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid \mathbf{s}_0)}{p(\tilde{\boldsymbol{\tau}}_{0:t}|\mathbf{s}_0, \mathbf{s}_{t^\star} = \mathbf{g})} d\mathbf{s}_{1:t} d\mathbf{a}_{0:t} \\ &= -\bar{\mathcal{F}}(\pi, \mathbf{s}_0, \mathbf{g}) + \log p(\mathbf{g}|\mathbf{s}_0), \end{aligned} \tag{B.7}$$

where

$$\begin{aligned} \bar{\mathcal{F}}(\pi, \mathbf{s}_0, \mathbf{g}) \doteq \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}_{0:t}|\mathbf{s}_0)} \Bigg[ & \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) + \log p(\mathbf{a}_t \mid \mathbf{s}_t) - \log \pi(\mathbf{a}_t \mid \mathbf{s}_t) \\ & + \sum_{t'=0}^{t-1} \log p(\mathbf{a}_{t'} \mid \mathbf{s}_{t'}) + \log p_d(\mathbf{s}_{t'+1} \mid \mathbf{s}_{t'}, \mathbf{a}_{t'}) \\ & - \log \pi(\mathbf{a}_{t'} \mid \mathbf{s}_{t'}) - \log q(\mathbf{s}_{t'+1} \mid \mathbf{s}_{t'}, \mathbf{a}_{t'}) \Bigg] \end{aligned} \tag{B.8}$$

and

$$\log p(\mathbf{g}|\mathbf{s}_0) = \log \int_{\mathcal{A}^t} \int_{\mathcal{S}^t} p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0, \mathbf{s}_{t^\star} = \mathbf{g}) d\mathbf{s}_{1:t} d\mathbf{a}_{0:t} \tag{B.9}$$

is a log-marginal likelihood. Following Haarnoja et al. [83], we assume that the variational distribution $q(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ matches the generative model, i.e., $q(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) = p_d(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$. We then get that

$$q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid \mathbf{s}_0) \doteq \pi(\mathbf{a}_t \mid \mathbf{s}_t) \prod_{t'=0}^{t-1} p_d(\mathbf{s}_{t'+1} \mid \mathbf{s}_{t'}, \mathbf{a}_{t'}) \pi(\mathbf{a}_{t'} \mid \mathbf{s}_{t'}), \tag{B.10}$$

we can simplify $\bar{\mathcal{F}}(\pi, \mathbf{s}_0, \mathbf{g})$ to

$$\bar{\mathcal{F}}(\pi, \mathbf{s}_0, \mathbf{g}) = \mathbb{E}_{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot|\mathbf{s}_0)} \left[ \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) + \sum_{t'=0}^{t} D_{\mathrm{KL}} \left( \pi(\cdot \mid \mathbf{s}_{t'}) \middle\| p(\cdot \mid \mathbf{s}_{t'}) \right) \right]. \tag{B.11}$$

Since $\log p(\mathbf{g}|\mathbf{s}_0)$ is constant in $\pi$, solving the variational optimization problem in Equation (B.5) is equivalent to maximizing the variational objective with respect to $\pi \in \Pi$, where $\Pi$ is a family of policy distributions. □

**Corollary 1** (Fixed-Time Outcome-Driven Reward Function)**.** *The objective in Equation (2.8) corresponds to KL-regularized reinforcement learning with a time-varying reward function given by*

$$r(\mathbf{s}_{t'}, \mathbf{a}_{t'}, \mathbf{g}, t') \doteq \mathbb{I}\{t' = t\} \log p_d(\mathbf{g} \mid \mathbf{s}_{t'}, \mathbf{a}_{t'}).$$

*Proof.* Let

$$r(\mathbf{s}_{t'}, \mathbf{a}_{t'}, \mathbf{g}, t') \doteq \mathbb{I}\{t' = t\} \log p_d(\mathbf{g} \mid \mathbf{s}_{t'}, \mathbf{a}_{t'}) \tag{B.12}$$

and note that the objective

$$\bar{\mathcal{F}}(\pi, \mathbf{s}_0, \mathbf{g}) = \mathbb{E}_{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot|\mathbf{s}_0)} \left[ \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) + \sum_{t=0}^{t} D_{\mathrm{KL}}(\pi(\cdot \mid \mathbf{s}_t) \,\|\, p(\cdot \mid \mathbf{s}_t) \right] \tag{B.13}$$

can equivalently written as

$$\bar{\mathcal{F}}(\pi, \mathbf{s}_0, \mathbf{g}) = \mathbb{E}_{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot|\mathbf{s}_0)} \left[ \sum_{t=0}^{t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}, \mathbf{g}, t') + \sum_{t'=0}^{t} D_{\mathrm{KL}} \left( \pi(\cdot \mid \mathbf{s}_{t'}) \middle\| p(\cdot \mid \mathbf{s}_{t'}) \right) \right] \tag{B.14}$$

$$= \mathbb{E}_{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot|\mathbf{s}_0)} \left[ \sum_{t=0}^{t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}, \mathbf{g}, t') + D_{\mathrm{KL}} \left( \pi(\cdot \mid \mathbf{s}_{t'}) \middle\| p(\cdot \mid \mathbf{s}_{t'}) \right) \right], \tag{B.15}$$

which, as shown in Haarnoja et al. [83], can be written in the form of Equation (2.1). □

**Proposition 2** (Unknown-time Outcome-Driven Variational Objective)**.** *Let* $q_{\tilde{\boldsymbol{\tau}}_{0:T},T}(\tilde{\boldsymbol{\tau}}_{0:t}, t \mid \mathbf{s}_0) = q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0) q_T(t)$*, let* $q_T(t)$ *be a variational distribution defined on* $t \in \mathbb{N}_0$*, and let* $q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0)$ *be as defined in Equation (2.7). Then, given any initial state* $\mathbf{s}_0$ *and outcome* $\mathbf{g}$*, we have that*

$$D_{\mathrm{KL}}(q_{\tilde{\boldsymbol{\tau}}_{0:T},T}(\cdot \mid \mathbf{s}_0) \,\|\, p_{\tilde{\boldsymbol{\tau}}_{0:T},T}(\cdot \mid \mathbf{s}_0, \mathbf{s}_{T^\star} = \mathbf{g})) = \log p(\mathbf{g}|\mathbf{s}_0) - \mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}), \tag{B.16}$$

*where*

$$\begin{aligned} \mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) \doteq \sum_{t=0}^{\infty} q_T(t \mid \mathbf{s}_0) \cdot \mathbb{E}_{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t}|t,\mathbf{s}_0)} \Big[ \\ \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) - D_{\mathrm{KL}}(q_{\tilde{\boldsymbol{\tau}}_{0:T},T}(\cdot \mid \mathbf{s}_0) \,||\, p_{\tilde{\boldsymbol{\tau}}_{0:T},T}(\cdot \mid \mathbf{s}_0)) \Big] \end{aligned} \tag{B.17}$$

*and* $\log p(\mathbf{g} \mid \mathbf{s}_0)$ *is constant in* $\pi$ *and* $q_T$*.*

*Proof.* In general, solving the variational problem

$$\min_{q \in \mathcal{Q}} D_{\mathrm{KL}} \left( q_{\tilde{\boldsymbol{\tau}}_{0:T},T}(\cdot \mid \mathbf{s}_0) \middle\| p_{\tilde{\boldsymbol{\tau}}_{0:T},T}(\cdot \mid \mathbf{s}_0, \mathbf{s}_{T^\star} = \mathbf{g}) \right) \tag{B.18}$$

from Section 2.3.3 in closed form is challenging, but as in the fixed-time setting, we can take advantage of the fact that, by choosing a variational family parameterized by

$$q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0) \doteq \pi(\mathbf{a}_t \mid \mathbf{s}_t) \prod_{t'=0}^{t-1} p_d(\mathbf{s}_{t'+1} \mid \mathbf{s}_{t'}, \mathbf{a}_{t'}) \, \pi(\mathbf{a}_{t'} \mid \mathbf{s}_{t'}), \tag{B.19}$$

with $\pi \in \Pi$, we can follow the same steps as in the proof for Proposition 1 and show that given any initial state $\mathbf{s}_0$ and outcome $\mathbf{g}$,

$$D_{\mathrm{KL}} \left( q_{\tilde{\boldsymbol{\tau}}_{0:T},T}(\cdot \mid \mathbf{s}_0) \middle\| p_{\tilde{\boldsymbol{\tau}}_{0:T},T}(\cdot \mid \mathbf{s}_0, \mathbf{s}_{T^\star} = \mathbf{g}) \right) = \log p(\mathbf{g} \mid \mathbf{s}_0) - \mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}), \tag{B.20}$$

where

$$\begin{aligned} \mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) \doteq \sum_{t=0}^{\infty} q_T(t) \mathbb{E}_{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0)} \Bigg[ & \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) \\ & - D_{\mathrm{KL}} \left( q_{\tilde{\boldsymbol{\tau}}_{0:T},T}(\cdot \mid \mathbf{s}_0) \middle\| p_{\tilde{\boldsymbol{\tau}}_{0:T},T}(\cdot \mid \mathbf{s}_0) \right) \Bigg], \end{aligned} \tag{B.21}$$

where $q(\tilde{\boldsymbol{\tau}}_{0:t}, t | \mathbf{s}_0) \doteq q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0) q_T(t)$, and hence, solving the variational problem in Equation (2.10) is equivalent to maximizing $\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g})$ with respect to $\pi$ and $q_T$. □

### B.1.2 Derivation of Recursive Variational Objective

**Proposition 4** (Factorized Unknown-Time Outcome-Driven Variational Objective)**.** *Let* $q_{\tilde{\boldsymbol{\tau}}_{0:T},T}(\tilde{\boldsymbol{\tau}}_{0:T}, T \mid \mathbf{s}_0) = q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0) q_T(t)$, *let* $q_T(t) = q_{\Delta_{t+1}}(\Delta_{t+1} = 1) \prod_{t'=1}^{t} q_{\Delta_{t'}}(\Delta_{t'} = 0)$ *be a variational distribution defined on* $t \in \mathbb{N}_0$, *and let* $q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0)$ *be as defined in Equation (2.7). Then, given any initial state* $\mathbf{s}_0$ *and outcome* $\mathbf{g}$, *Equation (B.17) can be rewritten as*

$$\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) = \mathbb{E}_{q(\tilde{\boldsymbol{\tau}} | \mathbf{s}_0)} \left[ \sum_{t=0}^{\infty} \left( \prod_{t'=1}^{t} q_{\Delta_{t'}}(\Delta_{t'} = 0) \right) \left( r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) - D_{\mathrm{KL}}(\pi(\cdot \mid \mathbf{s}_t) \,\|\, p(\cdot \mid \mathbf{s}_t)) \right) \right] \tag{B.22}$$

*Proof.* Consider the variational objective $\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g})$ in Equation (B.17):

$$\begin{aligned}
&\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) \\
&= \sum_{t=0}^{\infty} q_T(t) \mathbb{E}_{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} | t, \mathbf{s}_0)} \Big[ \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) - D_{\mathrm{KL}} \left( q_{\tilde{\boldsymbol{\tau}}_{0:T}, T}(\cdot \mid \mathbf{s}_0) \middle\| p_{\tilde{\boldsymbol{\tau}}_{0:T}, T}(\cdot \mid \mathbf{s}_0) \right) \Big] && \text{(B.23)} \\
&= \sum_{t=0}^{\infty} q_T(t \mid \mathbf{s}_0) \mathbb{E}_{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} | t, \mathbf{s}_0)} \left[ \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) - \log \frac{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0) q_T(\cdot \mid \mathbf{s}_0)(t)}{p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0) p_T(\cdot \mid \mathbf{s}_0)} d\tilde{\boldsymbol{\tau}}_{0:t} \right] && \text{(B.24)} \\
&= \sum_{t=0}^{\infty} q_T(t \mid \mathbf{s}_0) \mathbb{E}_{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} | t, \mathbf{s}_0)} \left[ \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) - \log \frac{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0)}{p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0)} \right] \\
&\qquad - \sum_{t=0}^{\infty} q_T(t \mid \mathbf{s}_0) \log \frac{q_T(t \mid \mathbf{s}_0)}{q_T(t \mid \mathbf{s}_0)}. && \text{(B.25)}
\end{aligned}$$

Noting that $\sum_{t=0}^{\infty} q_T(t \mid \mathbf{s}_0) \log \frac{q_T(t|\mathbf{s}_0)}{q_T(t|\mathbf{s}_0)} = D_{\mathrm{KL}} \left( q_T(\cdot \mid \mathbf{s}_0) \middle\| p_T(\cdot \mid \mathbf{s}_0) \right)$, we can write

$$\begin{aligned}
&\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) && \text{(B.26)} \\
&= \sum_{t=0}^{\infty} q_T(t \mid \mathbf{s}_0) \mathbb{E}_{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} | t, \mathbf{s}_0)} \left[ \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) - \log \frac{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0)}{p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0)} \right] \\
&\qquad - D_{\mathrm{KL}} \left( q_T(\cdot \mid \mathbf{s}_0) \middle\| p_T(\cdot \mid \mathbf{s}_0) \right) && \text{(B.27)} \\
&= \sum_{t=0}^{\infty} q_T(t \mid \mathbf{s}_0) \mathbb{E}_{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} | t, \mathbf{s}_0)} \Big[ \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) \Big] \\
&\qquad - \sum_{t=0}^{\infty} q_T(t \mid \mathbf{s}_0) \mathbb{E}_{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} | t, \mathbf{s}_0)} \left[ \log \frac{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0)}{p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0)} \right] \\
&\qquad - D_{\mathrm{KL}} \left( q_T(\cdot \mid \mathbf{s}_0) \middle\| p_T(\cdot \mid \mathbf{s}_0) \right). && \text{(B.28)}
\end{aligned}$$

Further noting that for an infinite-horizon trajectory distribution

$$q(\tilde{\boldsymbol{\tau}}_{t'} \mid \mathbf{s}_{t'}) \doteq \prod_{t=t'}^{\infty} p_d(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) \pi(\mathbf{a}_t \mid \mathbf{s}_t),$$

a trajectory realization $\tilde{\boldsymbol{\tau}}_{t+1} \doteq \{\boldsymbol{\tau}_{t'}\}_{t'=t+1}^{\infty}$, and any joint probability density $f(\mathbf{s}_t, \mathbf{a}_t)$,

$$\sum_{t=0}^{\infty} q_T(t \mid \mathbf{s}_0) \mathbb{E}_{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0)} \Big[ f(\mathbf{s}_t, \mathbf{a}_t) \Big] \tag{B.29}$$

$$= \sum_{t=0}^{\infty} \left( \mathbb{E}_{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0)} \Big[ q_T(t \mid \mathbf{s}_0) f(\mathbf{s}_t, \mathbf{a}_t) \Big] \cdot \underbrace{\left( \int q(\tilde{\boldsymbol{\tau}}_{t+1} \mid \mathbf{s}_0) d\tilde{\boldsymbol{\tau}}_{t+1} \right)}_{=1} \right) \tag{B.30}$$

$$= \sum_{t=0}^{\infty} \left( \left( \int_{\mathcal{S}^t \times \mathcal{A}^t} q(\tilde{\boldsymbol{\tau}}_{0:t} \mid \mathbf{s}_0) q_T(t \mid \mathbf{s}_0) f(\mathbf{s}_t, \mathbf{a}_t) d\tilde{\boldsymbol{\tau}}_{0:t} \right) \cdot \underbrace{\left( \int q(\tilde{\boldsymbol{\tau}}_{t+1} \mid \mathbf{s}_0) d\tilde{\boldsymbol{\tau}}_{t+1} \right)}_{=1} \right) \tag{B.31}$$

$$= \sum_{t=0}^{\infty} \left( \int q(\tilde{\boldsymbol{\tau}}_{t+1} \mid \mathbf{s}_0) \left( \int_{\mathcal{S}^t \times \mathcal{A}^t} q(\tilde{\boldsymbol{\tau}}_{0:t} \mid \mathbf{s}_0) q_T(t \mid \mathbf{s}_0) f(\mathbf{s}_t, \mathbf{a}_t) d\tilde{\boldsymbol{\tau}}_{0:t} \right) d\tilde{\boldsymbol{\tau}}_{t+1} \right), \tag{B.32}$$

$$= \sum_{t=0}^{\infty} \int q(\tilde{\boldsymbol{\tau}}_0 \mid \mathbf{s}_0) q_T(t \mid \mathbf{s}_0) f(\mathbf{s}_t, \mathbf{a}_t) d\tilde{\boldsymbol{\tau}}_0 \tag{B.33}$$

$$= \int q(\tilde{\boldsymbol{\tau}}_0 \mid \mathbf{s}_0) \sum_{t=0}^{\infty} q_T(t \mid \mathbf{s}_0) f(\mathbf{s}_t, \mathbf{a}_t) d\tilde{\boldsymbol{\tau}}_0, \tag{B.34}$$

we can express Equation (B.28) in terms of the infinite-horizon state–action trajectory $q(\tilde{\boldsymbol{\tau}}_0 \mid \mathbf{s}_0) \doteq \prod_{t=0}^{\infty} p_d(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) \pi(\mathbf{a}_t \mid \mathbf{s}_t)$ as

$$\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) \tag{B.35}$$

$$\begin{aligned} &= \int q(\tilde{\boldsymbol{\tau}}_0 | \mathbf{s}_0) \sum_{t=0}^{\infty} q_T(t \mid \mathbf{s}_0) \log p(\mathbf{g} | \mathbf{s}_t, \mathbf{a}_t) d\tilde{\boldsymbol{\tau}} \\ &\qquad - \sum_{t=0}^{\infty} q_T(t \mid \mathbf{s}_0) D_{\mathrm{KL}} \left( q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t, \mathbf{s}_0) \middle\| p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t, \mathbf{s}_0) \right) \\ &\qquad - D_{\mathrm{KL}} \left( q_T(\cdot \mid \mathbf{s}_0) \middle\| p_T(\cdot \mid \mathbf{s}_0) \right) \end{aligned} \tag{B.36}$$

$$\begin{aligned} &= \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}_0 | \mathbf{s}_0)} \left[ \sum_{t=0}^{\infty} q_T(t \mid \mathbf{s}_0) \Big( \log p(\mathbf{g} | \mathbf{s}_t, \mathbf{a}_t) - D_{\mathrm{KL}} \left( q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t, \mathbf{s}_0) \middle\| p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t, \mathbf{s}_0) \right) \Big) \right] \\ &\qquad - D_{\mathrm{KL}} \left( q_T(\cdot \mid \mathbf{s}_0) \middle\| p_T(\cdot \mid \mathbf{s}_0) \right). \end{aligned} \tag{B.37}$$

Using Lemma 7 and the definition of $q_T(t \mid \mathbf{s}_0)$ in Equation (2.11), we can rewrite this

objective as

$$\begin{aligned}
&\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) \\
&\quad = \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}_0|\mathbf{s}_0)}\Bigg[\sum_{t=0}^{\infty}\Big(\prod_{t'=1}^{t} q_{\Delta_{t'}}(\Delta_{t'}=0)\Big) q_{\Delta_{t'}}(\Delta_{t'}=1)\Big(\log p(\mathbf{g}|\mathbf{s}_t,\mathbf{a}_t) \\
&\qquad\qquad - D_{\mathrm{KL}}\left(q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t,\mathbf{s}_0)\middle\|p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t,\mathbf{s}_0)\right)\Big)\Bigg] \\
&\qquad\qquad - \sum_{t=0}^{\infty}\Big(\prod_{t'=1}^{t} q_{\Delta_{t'}}(\Delta_{t'}=0)\Big) D_{\mathrm{KL}}(q_{\Delta_{t+1}} \,||\, p_{\Delta_{t+1}}) \\
&\quad = \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}_0|\mathbf{s}_0)}\Bigg[\sum_{t=0}^{\infty}\left(\prod_{t'=1}^{t} q(\Delta_{t'}=0)\right)\cdot\Bigg(q(\Delta_{t+1}=1) \\
&\qquad\qquad \big(\log p(\mathbf{g}|\mathbf{s}_t,\mathbf{a}_t) - D_{\mathrm{KL}}\left(q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t,\mathbf{s}_0)\middle\|p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t,\mathbf{s}_0)\right)\big) - D_{\mathrm{KL}}\left(q_{\Delta_{t+1}}\middle\|p_{\Delta_{t+1}}\right)\Bigg)\Bigg],
\end{aligned} \tag{B.38}$$

(B.39)

with

$$\begin{aligned}
D_{\mathrm{KL}}\left(q_{\Delta_{t+1}}\middle\|p_{\Delta_{t+1}}\right) &= q_{\Delta_{t+1}}(\Delta_{t+1}=0)\log\frac{q_{\Delta_{t+1}}(\Delta_{t+1}=0)}{p_{\Delta_{t+1}}(\Delta_{t+1}=0)} \\
&\quad + (1-q_{\Delta_{t+1}}(\Delta_{t+1}=0))\log\frac{1-q_{\Delta_{t+1}}(\Delta_{t+1}=0)}{1-p_{\Delta_{t+1}}(\Delta_{t+1}=0)}.
\end{aligned} \tag{B.40}$$

Next, to re-express $D_{\mathrm{KL}}\left(q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t,\mathbf{s}_0)\middle\|p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t,\mathbf{s}_0)\right)$ as a sum over Kullback-Leibler divergences between distributions over single action random variables, we note that

$$\begin{aligned}
D_{\mathrm{KL}}\left(q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t,\mathbf{s}_0)\middle\|p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\cdot \mid t,\mathbf{s}_0)\right) &= \int_{\mathcal{S}^t\times\mathcal{A}^t} q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t,\mathbf{s}_0)\log\frac{q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t,\mathbf{s}_0)}{p_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t,\mathbf{s}_0)} d\tilde{\boldsymbol{\tau}}_{0:t} && \text{(B.41)} \\
&= \int_{\mathcal{S}^t\times\mathcal{A}^t} q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t,\mathbf{s}_0)\log\frac{\prod_{t'=1}^{t}\pi(\mathbf{a}_{t'} \mid \mathbf{s}_{t'})}{\prod_{t'=1}^{t}p(\mathbf{a}_{t'} \mid \mathbf{s}_{t'})} d\tilde{\boldsymbol{\tau}}_{0:t} && \text{(B.42)} \\
&= \int_{\mathcal{S}^t\times\mathcal{A}^t} q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t,\mathbf{s}_0)\sum_{t'=0}^{t}\log\frac{\pi(\mathbf{a}_{t'} \mid \mathbf{s}_{t'})}{p(\mathbf{a}_{t'} \mid \mathbf{s}_{t'})} d\tilde{\boldsymbol{\tau}}_{0:t} && \text{(B.43)} \\
&= \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}|\mathbf{s}_0)}\left[\sum_{t'=0}^{t}\int_{\mathcal{A}}\pi(\mathbf{a}_{t'} \mid \mathbf{s}_{t'})\log\frac{\pi(\mathbf{a}_{t'} \mid \mathbf{s}_{t'})}{p(\mathbf{a}_{t'} \mid \mathbf{s}_{t'})} d\mathbf{a}_{t'}\right] && \text{(B.44)} \\
&= \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}|\mathbf{s}_0)}\left[\sum_{t'=0}^{t} D_{\mathrm{KL}}\left(\pi(\cdot \mid \mathbf{s}_{t'})\middle\|p(\cdot \mid \mathbf{s}_{t'})\right)\right], && \text{(B.45)}
\end{aligned}$$

where we have used the same marginalization trick as above to express the expression in terms of an infinite-horizon trajectory distribution, which allows us to express Equation (B.39) as

$$\begin{aligned}
&\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) \\
&\quad = \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}_0|\mathbf{s}_0)}\Bigg[\sum_{t=0}^{\infty}\left(\prod_{t'=1}^{t} q(\Delta_{t'}=0)\right) \\
&\qquad\qquad \cdot\left(q(\Delta_{t+1}=1)\left(\log p(\mathbf{g}|\mathbf{s}_t,\mathbf{a}_t) - \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}|\mathbf{s}_0)}\left[\sum_{t'=0}^{t} D_{\mathrm{KL}}\left(\pi(\cdot \mid \mathbf{s}_{t'})\middle\|p(\cdot \mid \mathbf{s}_{t'})\right)\right]\right)\right. \\
&\qquad\qquad \left.- D_{\mathrm{KL}}\left(q_{\Delta_{t+1}}\middle\|p_{\Delta_{t+1}}\right)\right)\Bigg].
\end{aligned} \tag{B.46}$$

Rearranging and dropping redundant expectation operators, we can now express the objective as

$$\begin{aligned}
&\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) \\
&\quad = \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}_0|\mathbf{s}_0)}\Bigg[\sum_{t=0}^{\infty}\left(\prod_{t'=1}^{t} q(\Delta_{t'}=0)\right) \\
&\qquad\qquad \cdot\left(q(\Delta_{t+1}=1)\left(\log p(\mathbf{g}|\mathbf{s}_t,\mathbf{a}_t) - \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}|\mathbf{s}_0)}\left[\sum_{t'=0}^{t} D_{\mathrm{KL}}\left(\pi(\cdot \mid \mathbf{s}_{t'})\middle\|p(\cdot \mid \mathbf{s}_{t'})\right)\right]\right)\right. \\
&\qquad\qquad \left.- D_{\mathrm{KL}}\left(q_{\Delta_{t+1}}\middle\|p_{\Delta_{t+1}}\right)\right)\Bigg].
\end{aligned} \tag{B.47}$$

$$\begin{aligned}
&= \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}_0|\mathbf{s}_0)}\left[\sum_{t=0}^{\infty}\left(\prod_{t'=1}^{t} q(\Delta_{t'}=0)q(\Delta_{t+1}=1)\right)\log p(\mathbf{g}|\mathbf{s}_t,\mathbf{a}_t) - D_{\mathrm{KL}}\left(q_{\Delta_{t+1}}\middle\|p_{\Delta_{t+1}}\right)\right] \\
&\qquad - \sum_{t=0}^{\infty}\underbrace{\left(\prod_{t'=1}^{t} q(\Delta_{t'}=0)q(\Delta_{t+1}=1)\right)}_{=q_T(t|\mathbf{s}_0)}\mathbb{E}_{q(\tilde{\boldsymbol{\tau}}|\mathbf{s}_0)}\left[\sum_{t'=0}^{t} D_{\mathrm{KL}}\left(\pi(\cdot \mid \mathbf{s}_{t'})\middle\|p(\cdot \mid \mathbf{s}_{t'})\right)\right],
\end{aligned} \tag{B.48}$$

whereupon we note that the negative term can be expressed as

$$\begin{aligned}
&\sum_{t=0}^{\infty} q_T(t \mid \mathbf{s}_0)\mathbb{E}_{q(\tilde{\boldsymbol{\tau}}|\mathbf{s}_0)}\left[\sum_{t'=0}^{t} D_{\mathrm{KL}}\left(\pi(\cdot \mid \mathbf{s}_{t'})\middle\|p(\cdot \mid \mathbf{s}_{t'})\right)\right] \\
&\qquad = \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}|\mathbf{s}_0)}\left[\sum_{t=0}^{\infty}\sum_{t'=0}^{t} q_T(t \mid \mathbf{s}_0) D_{\mathrm{KL}}\left(\pi(\cdot \mid \mathbf{s}_{t'})\middle\|p(\cdot \mid \mathbf{s}_{t'})\right)\right] && \text{(B.49)} \\
&\qquad = \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}|\mathbf{s}_0)}\left[\sum_{t=0}^{\infty} q(T \geq t) D_{\mathrm{KL}}\left(\pi(\cdot \mid \mathbf{s}_t)\middle\|p(\cdot \mid \mathbf{s}_t)\right)\right] && \text{(B.50)} \\
&\qquad = \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}|\mathbf{s}_0)}\left[\sum_{t=0}^{\infty} \underbrace{\left(\prod_{t'=1}^{t} q_{\Delta_{t'}}(\Delta_{t'}=0)\right)}_{\text{(by Lemma 4)}} D_{\mathrm{KL}}\left(\pi(\cdot \mid \mathbf{s}_t)\middle\|p(\cdot \mid \mathbf{s}_t)\right)\right], && \text{(B.51)}
\end{aligned}$$

where the second line follows from expanding the sums and regrouping terms. By substituting the expression in Equation (B.51) into Equation (B.48), we obtain an objective expressed entirely in terms of distributions over single-index random variables:

$$\begin{aligned}
&\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) \\
&\quad = \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}_0|\mathbf{s}_0)}\left[\sum_{t=0}^{\infty}\left(\prod_{t'=1}^{t} q_{\Delta_{t'}}(\Delta_{t'}=0)\right)\right. \\
&\qquad\qquad \cdot\left(q_{\Delta_{t+1}}(\Delta_{t+1}=1)\log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) - D_{\mathrm{KL}}\left(q_{\Delta_{t+1}}\middle\|p_{\Delta_{t+1}}\right)\right) && \text{(B.52)} \\
&\qquad\qquad \left. - D_{\mathrm{KL}}\left(\pi(\cdot \mid \mathbf{s}_t)\middle\|p(\cdot \mid \mathbf{s}_t)\right)\right] \\
&\quad = \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}_0|\mathbf{s}_0)}\left[\sum_{t=0}^{\infty}\left(\prod_{t'=1}^{t} q_{\Delta_{t'}}(\Delta_{t'}=0)\right)\left(r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) - D_{\mathrm{KL}}\left(\pi(\cdot \mid \mathbf{s}_t)\middle\|p(\cdot \mid \mathbf{s}_t)\right)\right)\right], && \text{(B.53)}
\end{aligned}$$

where we defined

$$r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) \doteq q_{\Delta_{t+1}}(\Delta_{t+1}=1)\log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) - D_{\mathrm{KL}}\left(q_{\Delta_{t+1}}\middle\|p_{\Delta_{t+1}}\right), \tag{B.54}$$

which concludes the proof. □

**Theorem 1** (Outcome-Driven Variational Inference)**.** *Let* $q_T(t)$ *and* $q_{\tilde{\boldsymbol{\tau}}_{0:t}}(\tilde{\boldsymbol{\tau}}_{0:t} \mid t, \mathbf{s}_0)$ *be as defined in Equation (2.7) and Equation (2.11), and define*

$$\begin{aligned}
V^\pi(\mathbf{s}_t, \mathbf{g}; q_T) &\doteq \mathbb{E}_{\pi(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T)] - D_{\mathrm{KL}}(\pi(\cdot \mid \mathbf{s}_t) \,\|\, p(\cdot \mid \mathbf{s}_t)), && \text{(B.55)} \\
Q^\pi(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T) &\doteq r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) + q(\Delta_{t+1}=0)\mathbb{E}_{p_d(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}[V^\pi(\mathbf{s}_{t+1}, \mathbf{g}; \pi, q_T)], && \text{(B.56)} \\
r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) &\doteq q_{\Delta_{t+1}}(\Delta_{t+1}=1)\log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) - D_{\mathrm{KL}}(q_{\Delta_{t+1}} \,\|\, p_{\Delta_{t+1}}). && \text{(B.57)}
\end{aligned}$$

*Then given any initial state* $\mathbf{s}_0$ *and outcome* $\mathbf{g}$,

$$D_{\mathrm{KL}}(q_{q\tilde{\boldsymbol{\tau}}_{0:t},T}(\cdot \mid \mathbf{s}_0) \,\|\, p_{\tilde{\boldsymbol{\tau}}_{0:t},T}(\cdot \mid \mathbf{s}_0, \mathbf{s}_{T^\star} = \mathbf{g})) = -\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) + C = -V^\pi(\mathbf{s}_0, \mathbf{g}; q_T) + C,$$

*where* $C \doteq \log p(\mathbf{g} \mid \mathbf{s}_0)$ *is independent of* $\pi$ *and* $q_T$*, and hence maximizing* $V^\pi(\mathbf{s}_0, \mathbf{g}; \pi, q_T)$ *is equivalent to minimizing* Equation (2.10).

*Proof.* Consider the objective derived in Proposition 4 (Factorized Unknown-Time Outcome-Driven Variational Objective),

$$\begin{aligned}
&\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) \\
&\quad = \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}_0 \mid \mathbf{s}_0)} \Bigg[ \sum_{t=0}^{\infty} \left( \prod_{t'=1}^{t} q_{\Delta_{t'}}(\Delta_{t'} = 0) \right) \\
&\qquad\qquad \cdot \underbrace{\left( q_{\Delta_{t+1}}(\Delta_{t+1} = 1) \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) - D_{\mathrm{KL}} \left( q_{\Delta_{t+1}} \middle\| p_{\Delta_{t+1}} \right) \right)}_{\doteq\, r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta)} \\
&\qquad\qquad - D_{\mathrm{KL}} \left( \pi(\mathbf{a}_t \mid \mathbf{s}_t) \middle\| p(\mathbf{a}_t \mid \mathbf{s}_t) \right) \Bigg],
\end{aligned} \tag{B.58}$$

and recall that, by Proposition 2 (Unknown-time Outcome-Driven Variational Objective),

$$D_{\mathrm{KL}}(q_{q\tilde{\boldsymbol{\tau}}_{0:t},T}(\cdot \mid \mathbf{s}_0) \,\|\, p_{\tilde{\boldsymbol{\tau}}_{0:t},T}(\cdot \mid \mathbf{s}_0, \mathbf{s}_{T^\star} = \mathbf{g})) = -\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) + \log p(\mathbf{g} \mid \mathbf{s}_0). \tag{B.59}$$

Therefore, to prove the result that

$$D_{\mathrm{KL}}(q_{q\tilde{\boldsymbol{\tau}}_{0:t},T}(\cdot \mid \mathbf{s}_0) \,\|\, p_{\tilde{\boldsymbol{\tau}}_{0:t},T}(\cdot \mid \mathbf{s}_0, \mathbf{s}_{T^\star} = \mathbf{g})) = -V^\pi(\mathbf{s}_0, \mathbf{g}; q_T) + \log p(\mathbf{g} \mid \mathbf{s}_0),$$

we just need to show that $\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) = V^\pi(\mathbf{s}_0, \mathbf{g}; q_T)$ for $V^\pi(\mathbf{s}_0, \mathbf{g}; q_T)$ as defined in the theorem. To do so, we start from the objective $\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g})$ and and unroll it for $t = 0$:

$$\begin{aligned}
&\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) \\
&\quad = \mathbb{E}_{q(\tilde{\boldsymbol{\tau}}_0 \mid \mathbf{s}_0)} \left[ \sum_{t=0}^{\infty} \left( \prod_{t'=1}^{t} q_{\Delta_{t'}}(\Delta_{t'} = 0) \right) r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) - D_{\mathrm{KL}} \left( \pi(\mathbf{a}_t \mid \mathbf{s}_t) \middle\| p(\mathbf{a}_t \mid \mathbf{s}_t) \right) \right] \qquad \text{(B.60)} \\
&\quad = \mathbb{E}_{\pi(\mathbf{a}_0 \mid \mathbf{s}_0)} \Bigg[ r(\mathbf{s}_0, \mathbf{a}_0, \mathbf{g}; q_\Delta) + \mathbb{E}_{q(\boldsymbol{\tau}_1 \mid \mathbf{s}_0, \mathbf{a}_0)} \Big[ \sum_{t=1}^{\infty} \prod_{t'=1}^{t} q_{\Delta_{t'}}(\Delta_{t'} = 0) \Big( r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) \Big] \Bigg] \qquad \text{(B.61)} \\
&\qquad\qquad - D_{\mathrm{KL}} \left( \pi(\cdot \mid \mathbf{s}_t) \middle\| p(\cdot \mid \mathbf{s}_t) \right) \Big) - D_{\mathrm{KL}} \left( \pi(\cdot \mid \mathbf{s}_0) \middle\| p(\cdot \mid \mathbf{s}_0) \right).
\end{aligned}$$

With this expression at hand, let $r_0$ be shorthand for $r(\mathbf{s}_0, \mathbf{a}_0, \mathbf{g}; q_\Delta)$. We now define

$$\begin{aligned}
Q^\pi_{\mathrm{sum}}(\mathbf{s}_0, \mathbf{a}_0, \mathbf{g}; q_T) \doteq r_0 + \mathbb{E}_{q(\boldsymbol{\tau} \mid \mathbf{s}_0, \mathbf{a}_0)} \Bigg[ &\sum_{t=1}^{\infty} \prod_{t'=1}^{t} q_{\Delta_{t'}}(\Delta_{t'} = 0) \Big( r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) \\
&- D_{\mathrm{KL}} \left( \pi(\cdot \mid \mathbf{s}_t) \middle\| p(\cdot \mid \mathbf{s}_t) \right) \Big) \Bigg],
\end{aligned} \tag{B.62}$$

and note that

$$\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) = \mathbb{E}_{\pi(\mathbf{a}_0|\mathbf{s}_0)}[Q^\pi_{\text{sum}}(\mathbf{s}_0, \mathbf{a}_0, \mathbf{g}; q_T)] - D_{\text{KL}}(\pi(\cdot \mid \mathbf{s}_0) \,\|\, p(\cdot \mid \mathbf{s}_0)) = V^\pi(\mathbf{s}_0, \mathbf{g}; q_T),$$

as per the definition of $V^\pi(\mathbf{s}_0, \mathbf{g}; q_T)$. To prove the theorem from this intermediate result, we now have to show that $Q^\pi_{\text{sum}}(\mathbf{s}_0, \mathbf{a}_0, \mathbf{g}; q_T)$ as defined in Equation (B.62) can in fact be expressed recursively as $Q^\pi_{\text{sum}}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T) = Q^\pi(\mathbf{s}_0, \mathbf{a}_0, \mathbf{g}; q_T)$ with

$$Q^\pi(\mathbf{s}_0, \mathbf{a}_0, \mathbf{g}; q_T) = r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) + q(\Delta_{t+1} = 0)\mathbb{E}_{p_d(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}\left[V^\pi(\mathbf{s}_{t+1}, \mathbf{g}; \pi, q_T)\right]. \tag{B.63}$$

To see that this is the case, we begin by unrolling $Q^\pi(\mathbf{s}_0, \mathbf{a}_0, \mathbf{g}; q_T)$ for $t = 1$ to get

$$\begin{aligned}
&Q^\pi_{\text{sum}}(\mathbf{s}_0, \mathbf{a}_0, \mathbf{g}; q_T) \\
&= r_0 + \mathbb{E}_{q(\boldsymbol{\tau}_1|\mathbf{s}_0, \mathbf{a}_0)}\left[\sum_{t=1}^{\infty}\prod_{t'=1}^{t} q_{\Delta_{t'}}(\Delta_{t'} = 0)\Big(r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) - D_{\text{KL}}\left(\pi(\cdot \mid \mathbf{s}_t)\Big|\Big|p(\cdot \mid \mathbf{s}_t)\right)\Big)\right] && \text{(B.64)} \\
&= r_0 + \mathbb{E}_{p_d(\mathbf{s}_1|\mathbf{a}_0, \mathbf{a}_0)}\left[\mathbb{E}_{q(\boldsymbol{\tau}_1|\mathbf{s}_0, \mathbf{a}_0)}\left[\sum_{t=1}^{\infty}\prod_{t'=1}^{t} q_{\Delta_{t'}}(\Delta_{t'} = 0)\Big(r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta)\right.\right. \\
&\qquad \left.\left. - D_{\text{KL}}\left(\pi(\cdot \mid \mathbf{s}_t)\Big|\Big|p(\cdot \mid \mathbf{s}_t)\right)\Big)\right]\right] && \text{(B.65)} \\
&= r_0 + \mathbb{E}_{p_d(\mathbf{s}_1|\mathbf{a}_0, \mathbf{a}_0)}\left[\mathbb{E}_{\pi(\mathbf{a}_1|\mathbf{s}_1)}\left[q_{\Delta_1}(\Delta_1 = 0)\left(r(\mathbf{s}_1, \mathbf{a}_1, \mathbf{g}; q_\Delta) - D_{\text{KL}}\left(\pi(\cdot \mid \mathbf{s}_1)\Big|\Big|p(\cdot \mid \mathbf{s}_1)\right)\right)\right.\right. \\
&\quad \left.\left. + \mathbb{E}_{q(\boldsymbol{\tau}_2|\mathbf{s}_1, \mathbf{a}_1)}\left[\sum_{t=2}^{\infty}\prod_{t'=2}^{t} q_{\Delta_{t'}}(\Delta_{t'} = 0)\Big(r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) - D_{\text{KL}}\left(\pi(\cdot \mid \mathbf{s}_t)\Big|\Big|p(\cdot \mid \mathbf{s}_t)\right)\Big)\right]\right]\right]. && \text{(B.66)}
\end{aligned}$$

Next, note that we can rearrange this expression to obtain the recursive relationship

$$
\begin{aligned}
&Q^{\pi}_{\text{sum}}(\mathbf{s}_0, \mathbf{a}_0, \mathbf{g}; q_T) && \text{(B.67)}\\
&\quad = r_0 + q_{\Delta_1}(\Delta_1 = 0)\mathbb{E}_{p_d(\mathbf{s}_{0+1}|\mathbf{s}_0,\mathbf{a}_0)}\Bigg[ - D_{\text{KL}}\left(\pi(\cdot \mid \mathbf{s}_1)\middle\|p(\cdot \mid \mathbf{s}_1)\right)\\
&\qquad + \mathbb{E}_{\pi(\mathbf{a}_1|\mathbf{s}_1)}\Bigg[r(\mathbf{s}_1, \mathbf{a}_1, \mathbf{g}; q_\Delta) + \mathbb{E}\Bigg[\sum_{t=2}^{\infty}\left(\prod_{t'=2}^{t} q_{\Delta_{t'}}(\Delta_{t'} = 0)\right)\Big(r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) && \text{(B.68)}\\
&\qquad\qquad - D_{\text{KL}}\left(\pi(\cdot \mid \mathbf{s}_t)\middle\|p(\cdot \mid \mathbf{s})\right)\Big)\Bigg]\Bigg]\Bigg],\\
&\quad = r_0 + q_{\Delta_1}(\Delta_1 = 0)\mathbb{E}_{p_d(\mathbf{s}_{0+1}|\mathbf{s}_0,\mathbf{a}_0)}\Bigg[ - D_{\text{KL}}\left(\pi(\cdot \mid \mathbf{s}_1)\middle\|p(\cdot \mid \mathbf{s}_1)\right)\\
&\qquad + \mathbb{E}_{\pi(\mathbf{a}_1|\mathbf{s}_1)}\Big[Q^{\pi}_{\text{sum}}(\mathbf{s}_1, \mathbf{a}_1, \mathbf{g}; q_T)\Big]\Bigg], && \text{(B.69)}
\end{aligned}
$$

where the innermost expectation is taken with respect to $q(\boldsymbol{\tau}_2|\mathbf{s}_1, \mathbf{a}_1)$. With this result, we see that

$$
\begin{aligned}
&Q^{\pi}_{\text{sum}}(\mathbf{s}_0, \mathbf{a}_0, \mathbf{g}; q_T)\\
&\quad = r_0 + q_{\Delta_1}(\Delta_1 = 0)\mathbb{E}_{p_d(\mathbf{s}_{0+1}|\mathbf{s}_0,\mathbf{a}_0)}\Big[\mathbb{E}_{\pi(\mathbf{a}_1|\mathbf{s}_1)}\left[Q^{\pi}_{\text{sum}}(\mathbf{s}_1, \mathbf{a}_1, \mathbf{g}; q_T)\right] - D_{\text{KL}}\left(\pi(\cdot \mid \mathbf{s}_1)\middle\|p(\cdot \mid \mathbf{s}_1)\right)\Big] && \text{(B.70)}\\
&\quad = r_0 + q_{\Delta_1}(\Delta_1 = 0)\mathbb{E}_{p_d(\mathbf{s}_1|\mathbf{s},\mathbf{a})}\Big[V^{\pi}(\mathbf{s}_1, \mathbf{g}; q_T)\Big], && \text{(B.71)}
\end{aligned}
$$

for $V(\mathbf{s}_{t+1}, \mathbf{g}; q_T)$ as defined above, as desired. In other words, we have that

$$
\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) = \mathbb{E}_{\pi(\mathbf{a}_0|\mathbf{s}_0)}[Q^{\pi}_{\text{sum}}(\mathbf{s}_0, \mathbf{a}_0, \mathbf{g}; q_T)] - D_{\text{KL}}(\pi(\cdot \mid \mathbf{s}_0) \,\|\, p(\cdot \mid \mathbf{s}_0)) = V^{\pi}(\mathbf{s}_0, \mathbf{g}; q_T). \tag{B.72}
$$

Combining this result with Proposition 2 (Unknown-time Outcome-Driven Variational Objective) and Proposition 4 (Factorized Unknown-Time Outcome-Driven Variational Objective), we finally conclude that

$$
D_{\text{KL}}(q_{q_{\tilde{\boldsymbol{\tau}}_{0:t},T}}(\cdot \mid \mathbf{s}_0) \,\|\, p_{\tilde{\boldsymbol{\tau}}_{0:t},T}(\cdot \mid \mathbf{s}_0, \mathbf{s}_{T^\star} = \mathbf{g})) = -\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g}) + C = -V^{\pi}(\mathbf{s}_0, \mathbf{g}; q_T) + C, \tag{B.73}
$$

where $C \doteq \log p(\mathbf{g} \mid \mathbf{s}_0)$ is independent of $\pi$ and $q_T$. Hence, maximizing $V^{\pi}(\mathbf{s}_0, \mathbf{g}; \pi, q_T)$ is equivalent to minimizing the objective in Equation (2.10). This concludes the proof. □

**Corollary 3** (Simplified Outcome-Driven Variational Inference)**.** *Let $q_T = p_T$, assume that $p_T$ is a Geometric distribution with parameter $\gamma \in (0, 1)$, and let $p(\mathbf{a}_t|\mathbf{s}_t)$ be a uniform distribution.*

*Then the inference problem in Equation (2.10) of finding a goal-directed variational trajectory distribution simplifies to maximizing the following recursively defined variational objective with respect to $\pi$:*

$$\bar{V}^{\pi}(\mathbf{s}_0, \mathbf{g}; \gamma) \doteq \mathbb{E}_{\pi(\mathbf{a}_1|\mathbf{s}_1)}\left[Q(\mathbf{s}_1, \mathbf{a}_1, \mathbf{g}; \gamma)\right] - \mathcal{H}(\pi(\cdot \mid \mathbf{s}_0)), \tag{B.74}$$

*where*

$$\bar{Q}^{\pi}(\mathbf{s}_1, \mathbf{a}_1, \mathbf{g}; \gamma) \doteq (1-\gamma)\log p_d(\mathbf{g} \mid \mathbf{s}_1, \mathbf{a}_1) + \gamma \mathbb{E}_{p_d(\mathbf{s}_{0+1}|\mathbf{s}_0, \mathbf{a}_0)}\left[V(\mathbf{s}_2, \mathbf{g}; \gamma)\right] \tag{B.75}$$

*and $\mathcal{H}(\cdot)$ is the entropy functional.*

*Proof.* The result follows immediately when replacing $q_\Delta$ in Theorem 1 by $p_\Delta$ and noting that $D_{\mathrm{KL}}\left(p_\Delta \middle|\middle| p_\Delta\right) = 0$. □

### B.1.3 Derivation of Optimal Variational Posterior over $T$

**Proposition 3** (Optimal Variational Distribution over $T$)**.** *The optimal variational distribution $q_T^\star$ with respect to Equation (2.12) is defined recursively in terms of $q^\star_{\Delta_{t+1}}(\Delta_{t+1} = 0)$ for all $t \in \mathbb{N}_0$ by*

$$q^\star_{\Delta_{t+1}}(\Delta_{t+1} = 0; \pi, Q^\pi) = \sigma\Big(\mathbb{E}_{\pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})p_d(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}[Q^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}, \mathbf{g}; q_T)] \\ - \mathbb{E}_{\pi(\mathbf{a}_t|\mathbf{s}_t)}[\log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t)] + \sigma^{-1}\left(p_{\Delta_{t+1}}(\Delta_{t+1} = 0)\right)\Big)$$

*where $\sigma(\cdot)$ is the sigmoid function, that is, $\sigma(x) = \frac{1}{e^{-x}+1}$ and $\sigma^{-1}(x) = \log \frac{x}{1-x}$.*

*Proof.* Consider $\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g})$:

$$\mathcal{F}(\pi, q_T, \mathbf{s}_t, \mathbf{g}) = \mathbb{E}_{\pi(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T)] \tag{B.76}$$

$$= \mathbb{E}_{\pi(\mathbf{a}_t|\mathbf{s}_t)}[r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) + q_{\Delta_{t+1}}(\Delta_{t+1} = 0)\mathbb{E}\left[V(\mathbf{s}_{t+1}, \mathbf{g}; q_T)\right]]. \tag{B.77}$$

Since the variational objective $\mathcal{F}(\pi, q_T, \mathbf{s}_t, \mathbf{g})$ can be expressed recursively as

$$V^\pi(\mathbf{s}_t, \mathbf{g}; q_T) \doteq \mathbb{E}_{\pi(\mathbf{a}_t|\mathbf{s}_t)}\left[Q(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T)\right] - D_{\mathrm{KL}}\left(\pi(\cdot \mid \mathbf{s}_t) \middle|\middle| p(\cdot \mid \mathbf{s}_t)\right),$$

with

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T) = r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) + q_{\Delta_{t+1}}(\Delta_{t+1} = 0)\mathbb{E}_{p_d(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}\left[V^\pi(\mathbf{s}_{t+1}, \mathbf{g}; q_T)\right],$$
$$r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) = q_{\Delta_{t+1}}(\Delta_{t+1} = 1)\log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) - D_{\mathrm{KL}}\left(q_{\Delta_{t+1}} \middle|\middle| p_{\Delta_{t+1}}\right),$$

and since $D_{\mathrm{KL}}\left(q_{\Delta_{t+1}}\middle\|p_{\Delta_{t+1}}\right)$ is strictly convex in $q_{\Delta_{t+1}}(\Delta_{t+1}=0)$, we can find the globally optimal Bernoulli distribution parameters $q_{\Delta_{t+1}}(\Delta_{t+1}=0)$ for all $t\in\mathbb{N}_0$ recursively. That is, it is sufficient to solve the problem

$$q^{\star}_{\Delta_{t+1}}(\Delta_{t+1}=0) \doteq \underset{q_{\Delta_{t+1}}(\Delta_{t+1}=0)}{\arg\max}\ \{\mathcal{F}(\pi, q_T, \mathbf{s}_0, \mathbf{g})\} \tag{B.78}$$

$$= \underset{q_{\Delta_{t+1}}(\Delta_{t+1}=0)}{\arg\max}\ \left\{\mathcal{F}(\pi, q_{\Delta_1}, \ldots, q_{\Delta_{t+1}}, \ldots, \mathbf{s}_0, \mathbf{g})\right\} \tag{B.79}$$

for a fixed $t+1$. To do so, we take the derivative of $\mathcal{F}(\pi, q_{\Delta_1}, \ldots, q_{\Delta_{t+1}}, \ldots, \mathbf{s}_0, \mathbf{g})$, which—defined recursively—is given by

$$\mathbb{E}_{\pi(\mathbf{a}_t|\mathbf{s}_t)}[Q(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T] - D_{\mathrm{KL}}\left(\pi(\cdot \mid \mathbf{s}_t)\middle\|p(\cdot \mid \mathbf{s}_t)\right) \tag{B.80}$$

$$= \mathbb{E}_{\pi(\mathbf{a}_t|\mathbf{s}_t)}\left[r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta) + q_{\Delta_{t+1}}(\Delta_{t+1}=0)\mathbb{E}_{p_d(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}[V^{\pi}(\mathbf{s}_{t+1}, \mathbf{g}; q_T)]\right] \tag{B.81}$$

$$= -D_{\mathrm{KL}}\left(\pi(\cdot \mid \mathbf{s}_t)\middle\|p(\cdot \mid \mathbf{s}_t)\right) \tag{B.82}$$

$$= \mathbb{E}_{\pi(\mathbf{a}_t|\mathbf{s}_t)}\Bigg[q_{\Delta_{t+1}}(\Delta_{t+1}=1)\log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) - D_{\mathrm{KL}}\left(q_{\Delta_{t+1}}\middle\|p_{\Delta_{t+1}}\right) \tag{B.83}$$

$$+ q_{\Delta_{t+1}}(\Delta_{t+1}=0)\mathbb{E}_{p_d(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}[V^{\pi}(\mathbf{s}_{t+1}, \mathbf{g}; q_T)]\Bigg] - D_{\mathrm{KL}}\left(\pi(\cdot \mid \mathbf{s}_t)\middle\|p(\cdot \mid \mathbf{s}_t)\right) \tag{B.84}$$

$$= \mathbb{E}_{\pi(\mathbf{a}_t|\mathbf{s}_t)}\Bigg[(1 - q_{\Delta_{t+1}}(\Delta_{t+1}=0))\log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) - D_{\mathrm{KL}}\left(q_{\Delta_{t+1}}\middle\|p_{\Delta_{t+1}}\right) \tag{B.85}$$

$$+ q_{\Delta_{t+1}}(\Delta_{t+1}=0)\mathbb{E}_{p_d(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}[V^{\pi}(\mathbf{s}_{t+1}, \mathbf{g}; q_T)]\Bigg] - D_{\mathrm{KL}}\left(\pi(\cdot \mid \mathbf{s}_t)\middle\|p(\cdot \mid \mathbf{s}_t)\right), \tag{B.86}$$

with respect to $q_{\Delta_{t+1}}(\Delta_{t+1}=0)$ and set it to zero, which yields

$$0 = -\mathbb{E}_{\pi(\mathbf{a}_t|\mathbf{s}_t)}\left[\log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) + \mathbb{E}_{\pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})p_d(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}[Q^{\pi}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}, \mathbf{g}; q_T)]\right]$$
$$+ \log\frac{1 - q^{\star}_{\Delta_{t+1}}(\Delta_{t+1}=0)}{1 - p_{\Delta_{t+1}}(\Delta_{t+1}=0)} - \log\frac{q^{\star}_{\Delta_{t+1}}(\Delta_{t+1}=0)}{p_{\Delta_{t+1}}(\Delta_{t+1}=0)}. \tag{B.87}$$

Rearranging, we get

$$\frac{q^{\star}_{\Delta_{t+1}}(\Delta_{t+1}=0)}{1 - q^{\star}_{\Delta_{t+1}}(\Delta_{t+1}=0)} = \exp\Bigg(\mathbb{E}_{\pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})p_d(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)\pi(\mathbf{a}_t|\mathbf{s}_t)}[Q^{\pi}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}, \mathbf{g}; q_T)$$
$$- \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t)] + \log\frac{p_{\Delta_{t+1}}(\Delta_{t+1}=0)}{1 - p_{\Delta_{t+1}}(\Delta_{t+1}=0)}\Bigg), \tag{B.88}$$

where the $Q$-function depends on $q(\Delta_{t'})$ with $t' > t$, but not on $q^\star_{\Delta_{t+1}}(\Delta_{t+1} = 0)$. Solving for $q^\star_{\Delta_{t+1}}(\Delta_{t+1} = 0)$, we obtain

$$q^\star_{\Delta_{t+1}}(\Delta_{t+1} = 0) \tag{B.89}$$

$$= \frac{\exp(\mathbb{E}[Q^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}, \mathbf{g}; q_T) - \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t)] + \log \frac{p_{\Delta_{t+1}}(\Delta_{t+1}=0)}{1-p_{\Delta_{t+1}}(\Delta_{t+1}=0)})}{1 + \exp(\mathbb{E}[Q^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}, \mathbf{g}; q_T) - \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t)] + \log \frac{p_{\Delta_{t+1}}(\Delta_{t+1}=0)}{1-p_{\Delta_{t+1}}(\Delta_{t+1}=0)})} \tag{B.90}$$

$$= \sigma\Big(\mathbb{E}[Q^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}, \mathbf{g}; q_T)] - \mathbb{E}[\log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t)] + \sigma^{-1}\left(p_{\Delta_{t+1}}(\Delta_{t+1} = 0)\right)\Big), \tag{B.91}$$

where the expectation is with respect to $\pi(\mathbf{a}_{t+1} \mid \mathbf{s}_{t+1}) p_d(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) \pi(\mathbf{a}_t \mid \mathbf{s}_t)$ and where $\sigma(\cdot)$ is the sigmoid function with $\sigma(x) = \frac{1}{e^{-x}+1}$ and $\sigma^{-1}(x) = \log \frac{x}{1-x}$. This concludes the proof. □

**Remark 2.** *As can be seen from Proposition 3 (Optimal Variational Distribution over T), the optimal approximation to the posterior over $T$ trades off short-term rewards via $\mathbb{E}_{\pi(\mathbf{a}_t|\mathbf{s}_t)}[r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta)]$, long-term rewards via $\mathbb{E}_{\pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})p_d(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}[Q^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}, \mathbf{g}; q_T)]$, and the prior log-odds of not achieving the outcome at a given point in time conditioned on the outcome not having been achieved yet, $\frac{p_{\Delta_{t+1}}(\Delta_{t+1}=0)}{1-p_{\Delta_{t+1}}(\Delta_{t+1}=0)}$.*

**Corollary 2** (Equivalence to Soft Actor-Critic [83])**.** *Let $q_T = p_T$, assume that $p_T$ is a Geometric distribution with parameter $\gamma \in (0, 1)$, and let $p(\mathbf{a}_t \mid \mathbf{s}_t)$ be a uniform distribution. If the model of the state transition distribution is a Gibbs distribution,*

$$\tilde{p}_d(\mathbf{g}|\mathbf{s}_t, \mathbf{a}_t; \gamma) \doteq \frac{\exp\left(-\beta E(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g})\right)}{Z(\mathbf{s}_t, \mathbf{a}_t; \beta)}, \tag{B.92}$$

*with $\beta \doteq \log(1-\gamma)$, $Z(\mathbf{s}_t, \mathbf{a}_t; \beta) \doteq \int_{\mathcal{S}} \exp\left(-\beta E(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}')\right) d\mathbf{g}' < \infty$ and $E : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$, then for a fixed $\mathbf{g}$ and a reward function*

$$\tilde{r}(\mathbf{s}_t, \mathbf{a}_t) \doteq E(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}) + \log Z(\mathbf{s}_t, \mathbf{a}_t; \beta), \tag{B.93}$$

*the inference problem in Equation (2.10) of finding a goal-directed variational trajectory distribution simplifies to the infinite-horizon, discounted Soft Actor-Critic objective [83], with*

$$Q^\pi(\mathbf{s}, \mathbf{a}) \doteq \tilde{r}(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}\big[V^\pi(\mathbf{s}_1)\big], \tag{B.94}$$

*where the expectation is w.r.t. the true state transition distribution $p_d(\mathbf{s}_1|\mathbf{s}, \mathbf{a})$, and*

$$V^\pi(\mathbf{s}_1) \doteq \mathbb{E}_{\pi(\mathbf{a}_1|\mathbf{s}_1)}\left[Q^\pi(\mathbf{s}_1, \mathbf{a}_1)\right] + \mathcal{H}(\pi(\mathbf{a}_1 \mid \mathbf{s}_1)), \tag{B.95}$$

*where $\mathcal{H}(\cdot)$ is the entropy functional.*

*Proof.* The result follows immediately from the definition of $Q^\pi$ in Theorem 1. □

**Remark 3.** *Corollary 2 shows that the infinite-horizon, discounted Soft Actor-Critic algorithm can be derived entirely from first principles. In contrast, Haarnoja et al. [83] do not derive the discounted infinite-horizon objective from first principles, but instead include a discount factor post-hoc. Corollary 2 provides a probabilistic justification for this post-hoc objective.*

### B.1.4 Lemmas

**Lemma 3.** *Let* $q(T = t) \doteq q(T = t|T \geq t) \prod_{i=1}^{t} q(T \neq i-1|T \geq i-1)$ *be a discrete probability distribution with support* $\mathbb{N}_0$*. Then for any* $t \in \mathbb{N}_0$*, we have that*

$$q(T \geq t) = \sum_{i=t}^{\infty} q(T = i|T \geq i) \prod_{j=1}^{i} q(T \neq j-1|T \geq j-1) = \prod_{i=1}^{t} q(T \neq i-1|T \geq i-1). \tag{B.96}$$

*Proof.* We proof the statement by induction on $t$.

Base case: For $t = 0$, $q(T \geq 0) = 1$ by definition of the empty product.

Inductive case: Note that $q(T \leq t) = \prod_{i=1}^{t} q(T = i-1|T \geq i-1)$. Show that

$$q(T \geq t) = \prod_{i=1}^{t} q(T \neq i-1|T \geq i-1) \Longrightarrow q(T \geq t+1) = \prod_{i=1}^{t+1} q(T \neq i-1|T \geq i-1). \tag{B.97}$$

Consider $q(T \geq t+1) = \sum_{i=t+1}^{\infty} q(T = i|T \geq i) \prod_{j=1}^{i} q(T \neq j-1|T \geq j-1)$. To prove the inductive hypothesis, we need to show that the following equality is true:

$$\sum_{i=t+1}^{\infty} q(T = i|T \geq i) \prod_{j=1}^{i} q(T \neq j-1|T \geq j-1) = \prod_{i=1}^{t+1} q(T \neq i-1|T \geq i-1) \tag{B.98}$$

$$\begin{aligned} \Longleftrightarrow \sum_{i=t}^{\infty} q(T = i|T \geq i) \prod_{j=1}^{i} q(T \neq j-1|T \geq j-1) & \\ - q(T = t|T \geq t) \prod_{j=1}^{t} q(T \neq j-1|T \geq j-1) & \\ = q(T \neq t|T \geq t) \prod_{i=1}^{t} q(T \neq i-1|T \geq i-1). & \end{aligned} \tag{B.99}$$

By the inductive hypothesis,

$$q(T \geq t) = \sum_{i=t}^{\infty} q(T = i|T \geq i) \prod_{j=1}^{i} q(T \neq j-1|T \geq j-1) = \prod_{i=1}^{t} q(T \neq i-1|T \geq i-1), \tag{B.100}$$

and so

$$\text{Equation (B.99)} \iff \prod_{j=1}^{t} q(T \neq j | T \geq j) - q(T \neq t+1 | T \geq t+1) \prod_{j=1}^{t} q(T = j | T \geq j) \tag{B.101}$$

$$= q(T \neq t | T \geq t) \prod_{i=1}^{t} q(T \neq i-1 | T \geq i-1). \tag{B.102}$$

Factoring out $\prod_{i=1}^{t} q(T \neq i-1 | T \geq i-1)$, we get

$$\text{Equation (B.99)} \iff \prod_{j=1}^{t} q(T \neq j-1 | T \geq j-1) \underbrace{(1 - q(T = t | T \geq t))}_{=q(T \neq t | T \geq t)} \tag{B.103}$$

$$= q(T \neq t | T \geq t) \prod_{j=1}^{t} q(T = j-1 | T \geq j-1) \tag{B.104}$$

$$\iff q(T \neq t | T \geq t) \prod_{j=1}^{t} q(T \neq j-1 | T \geq j-1) \tag{B.105}$$

$$= q(T \neq t | T \geq t) \prod_{j=1}^{t} q(T \neq j-1 | T \geq j-1), \tag{B.106}$$

which proves the inductive hypothesis. □

**Lemma 4.** *Let $q_T(t)$ and $p_T(t)$ be discrete probability distributions with support $\mathbb{N}_0$, let $\Delta_t$ be a Bernoulli random variable, with success defined as $T = t+1$ given that $T \geq t$, and let $q_{\Delta_t}$ be a discrete probability distribution over $\Delta_t$ for $t \in \mathbb{N} \backslash \{0\}$, so that*

$$\begin{aligned} q_{\Delta_{t+1}}(\Delta_{t+1} = 0) &\doteq q(T \neq t \mid T \geq t) \\ q_{\Delta_{t+1}}(\Delta_{t+1} = 1) &\doteq q(T = t \mid T \geq t). \end{aligned} \tag{B.107}$$

*Then we can write $q(T = t) = q_{\Delta_{t+1}}(\Delta_{t+1} = 1) \prod_{i=1}^{t} q_{\Delta_i}(\Delta_i = 0)$ for any $t \in \mathbb{N}_0$ and have that*

$$q(T \geq t) = \sum_{i=t}^{\infty} q_{\Delta_{i+1}}(\Delta_{i+1} = 1) \prod_{j=1}^{i} q_{\Delta_j}(\Delta_j = 0) = \prod_{i=1}^{t} q_{\Delta_i}(\Delta_i = 0). \tag{B.108}$$

*Proof.* By Lemma 3, we have that for any $t \in \mathbb{N}_0$

$$q(T \geq t) = \sum_{i=t}^{\infty} q(T = i | T \geq i) \prod_{j=1}^{i} q(T \neq j-1 | T \geq j-1) = \prod_{i=1}^{t} q(T \neq i-1 | T \geq i-1). \tag{B.109}$$

The result follows by replacing $q(T = i | T \geq i)$ by $q_{\Delta_{i+1}}(\Delta_{i+1} = 1)$, $q(T \neq j-1 | T \geq j-1)$ by $q_{\Delta_j}(\Delta_j = 0)$, and $q(T \neq i-1 | T \geq i-1)$ by $q_{\Delta_i}(\Delta_i = 0)$. □

**Lemma 5.** *Let $q_T(t)$ and $p_T(t)$ be discrete probability distributions with support $\mathbb{N}_0$. Then for any $k \in \mathbb{N}_0$,*

$$\mathbb{E}_{t\sim q(T|T\geq k)}\left[\log \frac{q(T=t \mid T\geq k)}{p(T=t \mid T\geq k)}\right] = f(q,p,k) + q(T\neq k \mid T\geq k)\mathbb{E}_{t\sim q(T|T\geq k+1)}\left[\log \frac{q(T=t \mid T\geq k+1)}{p(T=t \mid T\geq k+1)}\right]. \tag{B.110}$$

*Proof.* Consider $\mathbb{E}_{t\sim q(T|T\geq k)}\left[\log \frac{q(T=t|T\geq k)}{p(T=t|T\geq k)}\right]$ and note that by the law of total expectation we can rewrite it as

$$\begin{aligned}
&\mathbb{E}_{t\sim q(T|T\geq k)}\left[\log \frac{q(T=t \mid T\geq k)}{p(T=t \mid T\geq k)}\right] \\
&= q(T=k \mid T\geq k)\mathbb{E}_{t\sim q(T|T=k)}\left[\log \frac{q(T=t \mid T\geq k)}{p(T=t \mid T\geq k)}\right] \\
&\qquad + q(T\neq k \mid T\geq k)\mathbb{E}_{t\sim q(T|T\geq k+1)}\left[\log \frac{q(T=t \mid T\geq k)}{p(T=t \mid T\geq k)}\right] && \text{(B.111)} \\
&= q(T=k \mid T\geq k)\log \frac{q(T=k \mid T\geq k)}{p(T=k \mid T\geq k)} && \text{(B.112)} \\
&\qquad + q(T\neq k \mid T\geq k)\mathbb{E}_{t\sim q(T|T\geq k+1)}\left[\log \frac{q(T=t \mid T\geq k)}{p(T=t \mid T\geq k)}\right]. && \text{(B.113)}
\end{aligned}$$

For all values of $T \geq k+1$, we have that

$$q(T=t \mid T\geq k) = q(T=t \mid T\geq k+1)q(T\neq k \mid T\geq k) \tag{B.114}$$
$$p(T=t \mid T\geq k) = p(T=t \mid T\geq k+1)p(T\neq k \mid T\geq k) \tag{B.115}$$

and so we can rewrite the expectation in Equation (B.112) as

$$\begin{aligned}
&\mathbb{E}_{t\sim q(T|T\geq k+1)}\left[\log \frac{q(T=t \mid T\geq k)}{p(T=t \mid T\geq k)}\right] \\
&= \mathbb{E}_{t\sim q(T|T\geq k+1)}\left[\log \frac{q(T=t \mid T\geq k)}{p(T=t \mid T\geq k)} + \log \frac{q(T\neq k \mid T\geq k)}{p(T\neq k \mid T\geq k)}\right] && \text{(B.116)} \\
&= \mathbb{E}_{t\sim q(T|T\geq k+1)}\left[\log \frac{q(T=t \mid T\geq k)}{p(T=t \mid T\geq k)}\right] + \log \frac{q(T\neq k \mid T\geq k)}{p(T\neq k \mid T\geq k)} && \text{(B.117)}
\end{aligned}$$

Combining Equation (B.117) with Equation (B.112), we have

$$\begin{aligned}
&\mathbb{E}_{t\sim q(T|T\geq k)}\left[\log\frac{q(T=t\mid T\geq k)}{p(T=t\mid T\geq k)}\right] \\
&\quad= \underbrace{q(T=k\mid T\geq k)\log\frac{q(T=k\mid T\geq k)}{p(T=k\mid T\geq k)}+q(T\neq k\mid T\geq k)\log\frac{q(T\neq k\mid T\geq k)}{p(T\neq k\mid T\geq k)}}_{\doteq f(q,p,k)} \\
&\qquad+q(T\neq k\mid T\geq k)\mathbb{E}_{t\sim q(T|T\geq k+1)}\left[\log\frac{q(T=t\mid T\geq k+1)}{p(T=t\mid T\geq k+1)}\right],
\end{aligned} \tag{B.118}$$

which concludes the proof. □

**Lemma 6.** *Let $q_T(t)$ and $p_T(t)$ be discrete probability distributions with support $\mathbb{N}_0$. Then the KL divergence from $q_T$ to $p_T$ can be written as*

$$D_{\mathrm{KL}}(q_T(\cdot\mid \mathbf{s}_0)\,||\,p_T(\cdot\mid \mathbf{s}_0))=\sum_{t=0}^{\infty}q(T\geq t)f(q_T,p_T,t) \tag{B.119}$$

*where $f(q_T,p_T,t)$ is shorthand for*

$$f(q_T,p_T,t)=q(T=t\mid T\geq t)\log\frac{q(T=t\mid T\geq t)}{p(T=t\mid T\geq t)}+q(T\neq t\mid T\geq t)\log\frac{q(T\neq t\mid T\geq t)}{p(T\neq t\mid T\geq t)}. \tag{B.120}$$

*Proof.* Note that $q(T=k)$ denotes the probability that the distribution $q$ assigns to the event $T=k$ and $q(T\geq m)$ denotes the tail probability, that is, $q(T\geq m)=\sum_{t=m}^{\infty}q(T=t)$. We will write $q(T|T\geq m)$ to denote the conditional distribution of $q$ given $T\geq m$, that is, $q(T=k|T\geq m)=\mathbb{1}[k\geq m]q(T=k)/q(T\geq m)$. We will use analogous notation for $p$.

By the definition of the KL divergence and using the fact that, since the support is lowerbounded by $T=0$, $q(T=0)=q(T=0\mid T\geq 0)$, we have

$$D_{\mathrm{KL}}\left(q_T(\cdot\mid \mathbf{s}_0)\middle\|p_T(\cdot\mid \mathbf{s}_0)\right)=\mathbb{E}_{t\sim q(T)}\left[\log\frac{q(T=t)}{p(T=t)}\right]=\mathbb{E}_{t\sim q(T|T\geq 0)}\left[\log\frac{q(T=t\mid T\geq 0)}{p(T=t\mid T\geq 0)}\right]. \tag{B.121}$$

Using Lemma 5 with $k = 0, 1, 2, 3, \ldots$, we can expand the above expression to get

$$
\begin{aligned}
&D_{\mathrm{KL}}\left(q_T(\cdot \mid \mathbf{s}_0)\middle\|p_T(\cdot \mid \mathbf{s}_0)\right) \\
&\quad= f(q_T, p_T, 0) + q(T \neq 0 \mid T \geq 0)\mathbb{E}_{t\sim q(T\mid T\geq 1)}\left[\log \frac{q(T=t \mid T\geq 1)}{p(T=t \mid T\geq 1)}\right] && \text{(B.122)} \\
&\quad= f(q, p, 0) + q(T \neq 0 \mid T \geq 1) f(q_T, p_T, 1) \\
&\qquad + q(T \neq 0 \mid T \geq 0) q(T \neq 1 \mid T \geq 1)\mathbb{E}_{t\sim q(T\mid T\geq 2)}\left[\log \frac{q(T=t \mid T\geq 2)}{p(T=t \mid T\geq 2)}\right] && \text{(B.123)} \\
&\quad= \underbrace{1}_{=q(T\geq 0)} \cdot f(q, p, 0) \\
&\qquad + \underbrace{q(T \neq 0 \mid T \geq 0)}_{=q(T\geq 1)} f(q, p, 1) \\
&\qquad + \underbrace{q(T \neq 0 \mid T \geq 0) q(T \neq 1 \mid T \geq 1)}_{=q(T\geq 2)} f(q_T, p_T, 2) \\
&\qquad + \underbrace{q(T \neq 0 \mid T \geq 0) q(T \neq 1 \mid T \geq 1) q(T \neq 2 \mid T \geq 2)}_{=q(T\geq 3)} \cdot \Bigg( && \text{(B.124)} \\
&\qquad\qquad \mathbb{E}_{t\sim q(T\mid T\geq 3)}\left[\log \frac{q(T=t \mid T\geq 3)}{p(T=t \mid T\geq 3)}\right] \Bigg) \\
&\quad= \sum_{t=0}^{\infty} q(T \geq t) f(q_T, p_T, t), && \text{(B.125)}
\end{aligned}
$$

where $f(q_T, p_T, t)$ is shorthand for

$$
f(q_T, p_T, t) = q(T = t \mid T \geq t) \log \frac{q(T = t \mid T \geq t)}{p(T = t \mid T \geq t)} + q(T \neq t \mid T \geq t) \log \frac{q(T \neq t \mid T \geq t)}{p(T \neq t \mid T \geq t)}. \tag{B.126}
$$

and we used the fact that, by Lemma 3,

$$
q(T \geq t) = \prod_{k=1}^{t} q(T \neq k-1 \mid T \geq k-1). \tag{B.127}
$$

This completes the proof. □

**Lemma 7.** *Let $q_T(t)$ and $p_T(t)$ be discrete probability distributions with support $\mathbb{N}_0$, let $\Delta_t$ be a Bernoulli random variable, with success defined as $T = t$ given that $T \geq t$, and let $q_{\Delta_t}$ and $p_{\Delta_t}$ be discrete probability distributions over $\Delta_t$ for $t \in \mathbb{N}_0 \backslash \{0\}$, so that*

$$
\begin{aligned}
q_{\Delta_{t+1}}(\Delta_{t+1} = 0) &\doteq q(T \neq t \mid T \geq t) \qquad q_{\Delta_{t+1}}(\Delta_{t+1} = 1) \doteq q(T = t \mid T \geq t) && \text{(B.128)} \\
p_{\Delta_{t+1}}(\Delta_{t+1} = 0) &\doteq p(T \neq t \mid T \geq t) \qquad p_{\Delta_{t+1}}(\Delta_{t+1} = 1) \doteq p(T = t \mid T \geq t). && \text{(B.129)}
\end{aligned}
$$

*Then the KL divergence from* $q_T(\cdot \mid \mathbf{s}_0)$ *to* $p_T(\cdot \mid \mathbf{s}_0)$ *can be written as*

$$D_{\mathrm{KL}}(q_T(\cdot \mid \mathbf{s}_0) \,||\, p_T(\cdot \mid \mathbf{s}_0)) = \sum_{t=0}^{\infty} \Big( \prod_{k=1}^{t} q_{\Delta_t}(\Delta_t = 0) \Big) D_{\mathrm{KL}}(q_{\Delta_{t+1}} \,||\, p_{\Delta_{t+1}}) \tag{B.130}$$

*Proof.* The result follows from Lemma 6, Equation (B.127), Equation (B.128), and the definition of $f$.

In detail, from Lemma 3, and Equation (B.128) we have that

$$q(T \geq t) = \prod_{k=1}^{t} q(T \neq k-1 \mid T \geq k-1) = \prod_{k=1}^{t} q_{\Delta_k}(\Delta_k = 0). \tag{B.131}$$

From the definition of $f(q_T, p_T, t)$, we have

$$f(q_T, p_T, t) = q(T = t \mid T \geq t) \log \frac{q(T = t \mid T \geq t)}{p(T = t \mid T \geq t)} + q(T \neq t \mid T \geq t) \log \frac{q(T \neq t \mid T \geq t)}{p(T \neq t \mid T \geq t)} \tag{B.132}$$

$$= q_{\Delta_{t+1}}(\Delta_{t+1} = 0) \log \frac{q_{\Delta_{t+1}}(\Delta_{t+1} = 0)}{p_{\Delta_{t+1}}(\Delta_{t+1} = 0)} + q(\Delta_{t+1} = 1) \log \frac{q_{\Delta_{t+1}}(\Delta_{t+1} = 1)}{p_{\Delta_{t+1}}(\Delta_{t+1} = 1)} \tag{B.133}$$

$$= D_{\mathrm{KL}} \left( q_{\Delta_{t+1}} \middle|\middle| p_{\Delta_{t+1}} \right). \tag{B.134}$$

Combining Equation (B.131), Equation (B.134), and Equation (B.119) completes the proof. □

## B.2 Proof of Outcome-Driven Policy Iteration Theorem

**Theorem 2** (Variational Outcome-Driven Policy Iteration)**.** *Assume* $|\mathcal{A}| < \infty$ *and that the MDP is ergodic.*

1. *Outcome-Driven Policy Evaluation (ODPE): Given policy* $\pi$ *and a function* $Q^0 : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$*, define* $Q^{i+1} = \mathcal{T}^\pi Q^i$*. Then the sequence* $Q^i$ *converges to the lower bound in Theorem 1.*

2. *Outcome-Driven Policy Improvement (ODPI): The policy that solves*

$$\pi^+ = \arg\max_{\pi' \in \Pi} \left\{ \mathbb{E}_{\pi'(\mathbf{a}_t \mid \mathbf{s}_t)} \left[ Q^\pi(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T) \right] - D_{\mathrm{KL}}(\pi'(\cdot \mid \mathbf{s}_t) \,||\, p(\cdot \mid \mathbf{s}_t)) \right\} \tag{B.135}$$

*and the variational distribution over $T$ recursively defined in terms of*

$$q^+(\Delta_{t+1}=0 \mid \mathbf{s}_0;\pi,Q^\pi)=\sigma\Big(\mathbb{E}_{\pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})p_d(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}[Q^\pi(\mathbf{s}_{t+1},\mathbf{a}_{t+1},\mathbf{g};q_T)] - \mathbb{E}_{\pi(\mathbf{a}_t|\mathbf{s}_t)}[\log p_d(\mathbf{g}\mid\mathbf{s}_t,\mathbf{a}_t)]+\sigma^{-1}\left(p_{\Delta_{t+1}}(\Delta_{t+1}=0)\right)\Big) \tag{B.136}$$

*improve the variational objective. In other words, $\mathcal{F}(\pi^+,q_T,\mathbf{s}_0)\geq\mathcal{F}(\pi,q_T,\mathbf{s}_0)$ and $\mathcal{F}(\pi,q_T^+,\mathbf{s}_0)\geq\mathcal{F}(\pi,q_T,\mathbf{s}_0)$ for all $\mathbf{s}_0\in\mathcal{S}$.*

3. *Alternating between ODPE and ODPI converges to a policy $\pi^\star$ and a variational distribution over $T$, $q_T^\star$, such that $Q^{\pi^\star}(\mathbf{s},\mathbf{a},\mathbf{g};q_T^\star)\geq Q^\pi(\mathbf{s},\mathbf{a},\mathbf{g};q_T)$ for all $(\pi,q_T)\in\Pi\times\mathcal{Q}_T$ and any $(\mathbf{s},\mathbf{a})\in\mathcal{S}\times\mathcal{A}$.*

*Proof.* Parts of this proof are adapted from the proof given in Haarnoja et al. [83], modified for the Bellman operator proposed in Definition 1.

1. Outcome-Driven Policy Evaluation (ODPE): Instead of absorbing the entropy term into the $Q$-function, we can define an entropy-augmented reward as

$$r^\pi(\mathbf{s}_t,\mathbf{a}_t,\mathbf{g};q_\Delta)\doteq q_{\Delta_{t+1}}(\Delta_{t+1}=1)\log p_d(\mathbf{g}\mid\mathbf{s}_t,\mathbf{a}_t)-D_{\mathrm{KL}}\left(q_{\Delta_{t+1}}\middle\|p_{\Delta_{t+1}}\right) + q_{\Delta_{t+1}}(\Delta_{t+1}=0)\mathbb{E}_{p_d(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}[D_{\mathrm{KL}}\left(\pi(\cdot\mid\mathbf{s}_{t+1})\middle\|p(\cdot\mid\mathbf{s}_{t+1})\right)]. \tag{B.137}$$

We can then write an update rule according to Definition 1 as

$$\tilde{Q}(\mathbf{s}_t,\mathbf{a}_t,\mathbf{g};q_T)\leftarrow r^\pi(\mathbf{s}_t,\mathbf{a}_t,\mathbf{g};q_\Delta)+q_{\Delta_{t+1}}(\Delta_{t+1}=0)\cdot\Big( \tag{B.138}$$

$$\mathbb{E}_{\pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})p_d(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}\left[\tilde{Q}(\mathbf{s}_{t+1},\mathbf{a}_{t+1},\mathbf{g};q_T)\right]\Big), \tag{B.139}$$

where $q_{\Delta_{t+1}}(\Delta_{t+1}=0)\leq 1$. This update is similar to a Bellman update [200], but with a discount factor given by $q_{\Delta_{t+1}}(\Delta_{t+1}=0)$. In general, this discount factor $q_{\Delta_{t+1}}(\Delta_{t+1}=0)$ can be computed dynamically based on the current state and action, such as in Equation (2.15). As discussed in White [219], this Bellman operator is still a contraction mapping so long as the Markov chain induced by the current policy is ergodic and there exists a state such that $q_{\Delta_{t+1}}(\Delta_{t+1}=0)<1$. The first condition is true by assumption. The second condition is true since $q_{\Delta_{t+1}}(\Delta_{t+1}=0)$ is given by Equation (2.15), which is always strictly between 0 and 1. Therefore, we apply convergence results for policy evaluation with transition-dependent discount factors [219] to this contraction mapping, and the result immediately follows.

2. Outcome-Driven Policy Improvement (ODPI): Let $\pi_{\text{old}}\in\Pi$ and let $Q^{\pi_{\text{old}}}$ and $V^{\pi_{\text{old}}}$ be the outcome-driven state and state-action value functions from Definition 1, let $q_T$ be

some variational distribution over $T$, and let $\pi_{\text{new}}$ be given by

$$\begin{aligned}\pi_{\text{new}}(\mathbf{a}_t|\mathbf{s}_t) &= \arg\max_{\pi' \in \Pi} \left\{ \mathbb{E}_{\pi'(\mathbf{a}_t|\mathbf{s}_t)} \left[ Q^{\pi_{\text{old}}}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T) \right] - D_{\text{KL}} \left( \pi'(\cdot \mid \mathbf{s}_t) \middle|\middle| p(\cdot \mid \mathbf{s}_t) \right) \right\} \\ &= \arg\max_{\pi' \in \Pi} \mathcal{J}_{\pi_{\text{old}}}(\pi'(\mathbf{a}_t, \mathbf{s}_t), q_T).\end{aligned} \tag{B.140}$$

Then, it must be true that $\mathcal{J}_{\pi_{\text{old}}}(\pi_{\text{old}}(\mathbf{a}_t|\mathbf{s}_t); q_T) \leq \mathcal{J}_{\pi_{\text{old}}}(\pi_{\text{new}}(\mathbf{a}_t|\mathbf{s}_t); q_T)$, since one could set $\pi_{\text{new}} = \pi_{\text{old}} \in \Pi$. Thus,

$$\begin{aligned}&\mathbb{E}_{\pi_{\text{new}}(\mathbf{a}_t|\mathbf{s}_t)} \left[ Q^{\pi_{\text{old}}}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T) \right] - D_{\text{KL}} \left( \pi_{\text{new}}(\cdot \mid \mathbf{s}_t) \middle|\middle| p(\cdot \mid \mathbf{s}_t) \right) \\ &\quad \geq \mathbb{E}_{\pi_{\text{old}}(\mathbf{a}_t|\mathbf{s}_t)} \left[ Q^{\pi_{\text{old}}}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T) \right] - D_{\text{KL}} \left( \pi_{\text{old}}(\cdot \mid \mathbf{s}_t) \middle|\middle| p(\cdot \mid \mathbf{s}_t) \right),\end{aligned} \tag{B.141}$$

and since

$$V^{\pi_{\text{old}}}(\mathbf{s}_t, \mathbf{g}; q_T) = \mathbb{E}_{\pi_{\text{old}}(\mathbf{a}_t|\mathbf{s}_t)} \left[ Q^{\pi_{\text{old}}}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T) \right] - D_{\text{KL}} \left( \pi_{\text{old}}(\cdot \mid \mathbf{s}_t) \middle|\middle| p(\cdot \mid \mathbf{s}_t) \right), \tag{B.142}$$

we get

$$\mathbb{E}_{\pi_{\text{new}}(\mathbf{a}_t|\mathbf{s}_t)} \left[ Q^{\pi_{\text{old}}}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T) \right] - D_{\text{KL}} \left( \pi_{\text{new}}(\cdot \mid \mathbf{s}_t) \middle|\middle| p(\cdot \mid \mathbf{s}_t) \right) \geq V^{\pi_{\text{old}}}(\mathbf{s}_t, \mathbf{g}; q_T). \tag{B.143}$$

We can now write the Bellman equation as

$$\begin{aligned}&Q^{\pi_{\text{old}}}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T) && \text{(B.144)} \\ &\quad = q_{\Delta_{t+1}}(\Delta_{t+1} = 1) \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) \\ &\qquad\qquad + q_{\Delta_{t+1}}(\Delta_{t+1} = 0) \mathbb{E}_{p_d(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} [V^{\pi_{\text{old}}}(\mathbf{s}_{t+1}, \mathbf{g}; q_T)] && \text{(B.145)} \\ &\quad \leq q_{\Delta_{t+1}}(\Delta_{t+1} = 1) \log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t) \\ &\qquad + q_{\Delta_{t+1}}(\Delta_{t+1} = 0) \mathbb{E}_{p(\mathbf{s}_{t'}|\mathbf{s}_t, \mathbf{a}_t)} [\mathbb{E}_{\pi_{\text{new}}(\mathbf{a}_{t'}|\mathbf{s}_{t'})} \left[ Q^{\pi_{\text{old}}}(\mathbf{s}_{t'}, \mathbf{a}_{t'}, \mathbf{g}; q_T) \right] \\ &\qquad\qquad - D_{\text{KL}} \left( \pi_{\text{new}}(\cdot \mid \mathbf{s}_{t'}) \middle|\middle| p(\cdot \mid \mathbf{s}_{t'}) \right)], && \text{(B.146)} \\ &\quad \vdots \\ &\quad \leq Q^{\pi_{\text{new}}}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_T) && \text{(B.147)}\end{aligned}$$

where we defined $t' \doteq t + 1$, repeatedly applied the Bellman backup operator defined in Definition 1 and used the bound in Equation (B.143). Convergence follows from Outcome-Driven Policy Evaluation above.

3. Locally Optimal Variational Outcome-Driven Policy Iteration: Define $\pi^i$ to be a policy at iteration $i$. By ODPI for a given $q_T$, the sequence of state-action value functions $\{Q^{\pi^i}(q_T)\}_{i=1}^{\infty}$ is monotonically increasing in $i$. Since the reward is finite and the negative

KL divergence is upper bounded by zero, $Q^{\pi}(q_T)$ is upper bounded for $\pi \in \Pi$ and the sequence $\{\pi^i\}_{i=1}^{\infty}$ converges to some $\pi^\star$. To see that $\pi^\star$ is an optimal policy, note that it must be the case that $\mathcal{J}_{\pi^\star}(\pi^\star(\mathbf{a}_t|\mathbf{s}_t); q_T) > \mathcal{J}_{\pi^\star}(\pi(\mathbf{a}_t \mid \mathbf{s}_t); q_T)$ for any $\pi \in \Pi$ with $\pi \neq \pi^\star$. By the argument used in ODPI above, it must be the case that the outcome-driven state-action value of the converged policy is higher than that of any other non-converged policy in $\Pi$, that is, $Q^{\pi^\star}(\mathbf{s}_t, \mathbf{a}_t; q_T) > Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t; q_T)$ for all $\pi \in \Pi$ and any $q_T^i \in \mathcal{Q}_T$ and $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$. Therefore, given $q_T$, $\pi^\star$ must be optimal in $\Pi$, which concludes the proof.

4. Globally Optimal Variational Outcome-Driven Policy Iteration: Let $\pi^i$ be a policy and let $q_T^i$ be variational distributions over $T$ at iteration $i$. By Locally Optimal Variational Outcome-Driven Policy Iteration, for a *fixed* $q_T^i$ with $q_T^i = q_T^j \forall i, j \in \mathbb{N}_0$, the sequence of $\{(\pi^i, q_T^i)\}_{i=1}^{\infty}$ increases the objective Equation (B.16) at each iteration and converges to a stationary point in $\pi^i$, where $Q^{\pi^\star}(\mathbf{s}_t, \mathbf{a}_t; q_T^i) > Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t; q_T^i)$ for all $\pi \in \Pi$ and any $q_T^i \in \mathcal{Q}_T$ and $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$. Since the objective in Equation (B.16) is concave in $q_T$, it must be the case that for, $q_T^{\star^i} \in \mathcal{Q}_T$, the optimal variational distribution over $T$ at iteration $i$, defined recursively by

$$q^{\star^i}(\Delta_{t+1} = 0; \pi^i, Q^{\pi^i}) = \sigma\Big(\mathbb{E}_{\pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})p_d(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}[Q^{\pi^i}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}, \mathbf{g}; q_T(\pi^i, Q^{\pi^i}))] \\ - \mathbb{E}_{\pi(\mathbf{a}_t|\mathbf{s}_t)}[\log p_d(\mathbf{g} \mid \mathbf{s}_t, \mathbf{a}_t)] + \sigma^{-1}(p_{\Delta_{t+1}}(\Delta_{t+1} = 0)\Big),$$

for $t \in \mathbb{N}_0$, $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t; q_T^\star) > Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t; q_T)$ for all $\pi \in \Pi$ and any $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$. Note that $q_T$ is defined implicitly in terms of $\pi^i$ and $Q^{\pi^i}$, that is, the optimal variational distribution over $T$ at iteration $i$ is defined as a function of the policy and $Q$-function at iteration $i$. Hence, it must then be true that for $Q^{\pi^\star}(\mathbf{s}_t, \mathbf{a}_t; q_T^\star) > Q^{\pi^\star}(\mathbf{s}_t, \mathbf{a}_t; q_T)$ for all $q_T^\star(\pi^\star, Q^{\pi^\star}) \in \mathcal{Q}_T$ and for any $\pi^\star \in \Pi$ and $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$. In other words, for an optimal policy and corresponding $Q$-function, there exists an optimal variational distribution over $T$ that maximizes the $Q$-function, given the optimal policy. Repeating locally optimal variational outcome-driven policy iteration under the new variational distribution $q_T^\star(\pi^\star, Q^{\pi^\star})$ will yield an optimal policy $\pi^{\star\star}$ and computing the corresponding optimal variational distribution, $q_T^{\star\star}(\pi^{\star\star}, Q^{\pi^{\star\star}})$ will further increase the variational objective such that for $\pi^{\star\star}) \in \Pi$ and $q_T^{\star\star}(\pi^{\star\star}, Q^{\pi^{\star\star}}) \in \mathcal{Q}_T$, we have that

$$Q^{\pi^{\star\star}}(\mathbf{s}_t, \mathbf{a}_t; q_T^{\star\star}) > Q^{\pi^{\star\star}}(\mathbf{s}_t, \mathbf{a}_t; q_T^\star) > Q^{\pi^\star}(\mathbf{s}_t, \mathbf{a}_t; q_T^\star) > Q^{\pi^\star}(\mathbf{s}_t, \mathbf{a}_t; q_T) \tag{B.148}$$

for any $\pi^\star \in \Pi$ and $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$. Hence, global optimal variational outcome-driven policy iteration increases the variational objective at every step. Since the objective is upper bounded (by virtue of the rewards being finite and the negative KL divergence being upper bounded by zero) and the sequence of $\{(\pi^i, q_T^i)\}_{i=1}^{\infty}$ increases the objective Equation (B.16) at each iteration, by the monotone convergence theorem, the objective value converges to a supremum and since the objective function is concave the supremum is unique. Hence, since the supremum is unique and obtained via

global optimal variational outcome–driven policy iteration on $(\pi, q_T) \in \Pi \times \mathcal{Q}_T$, the sequence of $\{(\pi^i, q_T^i)\}_{i=1}^{\infty}$ converges to a unique stationary point $(\pi^\star, q_T^\star) \in \Pi \times \mathcal{Q}_T$, where $Q^{\pi^\star}(\mathbf{s}_t, \mathbf{a}_t; q_T^\star) > Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t; q_T^i)$ for all $\pi \in \Pi$ and any $q_T^i \in \mathcal{Q}_T$ and $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$.

□

**Corollary 3.** *3Optimality of Variational Outcome Driven Policy Iteration Variational Outcome-Driven Policy Iteration on $(\pi, q_T) \in \Pi \times \mathcal{Q}_T$ results in an optimal policy at least as good or better than any optimal policy attainable from policy iteration on $\pi \in \Pi$ alone.*

**Remark 4.** *The convergence proof of ODPE assumes a transition-dependent discount factor [219], because the variational distribution used in Equation (2.15) depends on the next state and action as well as on the desired outcome.*

## B.3 Additional Experiments

**Description of Figure 2.1** We implemented a tabular version of ODAC and applied it to the 2D environment shown in Figure 2.3a. We discretize the environment into an $8 \times 8$ grid of states. The action correspond to moving up, down, left, or right. If probability $1 - \epsilon$, this action is taken. If the agent runs into a wall or boundary, the agent stays in its current state. With probability $\epsilon = 0.1$, the commanded action is ignored and a neighboring state grid (including the current state) is uniformly sampled as the next state. The policy and $Q$-function are represented with look-up tables and randomly initialized. The entropy reward is weighted by 0.01 and the time prior $p(T)$ is geometric with parameter 0.5. The dynamics model, $p_d^{(0)}$ is initialized to give a uniform probability to each states for every state and action. Each iteration, we simulate data collection by updating the dynamics model with the running average update

$$p_d^{(t+1)} = 0.99 p_d^{(t)} + 0.01 p_d$$

where $p_d$ is the true dynamics and update the policy and $Q$-function according to Equation (2.18) and Equation (2.16), respectively. Figure 2.1 shows that, in contrast to the binary-reward setting, the learned reward provides shaping for the policy, which solves the task within 100 iterations.

**Full Ablation Results** We show the full ablation learning curves in Figure B.1. We see that ODAC consistently performs the best, and that ODAC with a fixed model also performs well. However, on a few tasks, and in particular the Fetch Push and Sawyer Faucet tasks, we see that using a fixed $q_T$ hurts the performance, suggesting that our derived formula in Equation (2.15) results in better empirical performance.

**Comparison to methods with oracle goal sampling.** To demonstrate the impact of sampling the desired outcome $\mathbf{g}$ during exploration, we evaluate the methods on the Fetch task when using oracle goal sampling, in which the goals are sampled uniformly from the set of possible desired outcomes. As shown in Figure B.2, we see that the performances of UVD and ODAC are similar and that both outperform other methods. The large drop in performance when the desired outcome $\mathbf{g}$ is fixed may be due to the fact that uniformly sampling $\mathbf{g}$ implicitly provides a curriculum for learning. For example, in the Box 2D environment, goal states sampled above the box can train the agent to move around the obstacle, making it easier to learn how to reach the other side of the box. Without this guidance, prior methods often "got stuck" on the other side of the box. In contrast, ODAC consistently performs well in this more challenging setting, suggesting that the log-likelihood signal provides good guidance to the policy.

As shown in Figure 2.4, ODAC performs well on both this setting and the harder setting where the desired outcome $\mathbf{g}$ was fixed during exploration, suggesting that ODAC does not rely as heavily on the uniform sampling of $\mathbf{g}$ to learn a good policy than do other methods.

**Comparison to model-based planning.** ODAC learns a dynamics model but does not use it for planning and instead relies on the derived Bellman updates to obtain a policy. However, a natural question is whether or not the method would benefit from using this model to perform model-based planning, as in Janner et al. [102]. We assess this by comparing ODAC with model-based baseline that uses a 1-step look-ahead. In particular, we follow the training procedure in Janner et al. [102] with $k = 1$. To ensure a fair comparison, we use the exact same dynamics model architecture as in ODAC and match the update-to-environment step ratio to be 4-to-1 for both methods. Table B.1 shows the final distance to the goal (best results in bold). Using the same dynamics model, ODAC, which does not use the dynamics model to perform planning and only uses it to compute rewards, outperforms the model-based planning method. While a better model might lead to better performance for the model-based baseline, these results suggest that ODAC is not sensitive to model quality to the same degree as model-based planning methods.

**Reward Visualization** We visualize the reward for the Box 2D environment in Figure B.4. We see that over the course of training, the reward function initially flattens out near $\mathbf{g}$, making learning easier by encouraging the policy to focus on moving just out of the top left corner of the environment. Later in training (around 16,000 steps), the policy learns to move out of the top left corner, and we see that the reward changes to have a stronger reward gradient near $\mathbf{g}$. We also note that the reward are much more negative for being far $\mathbf{g}$ at the end of training: the top left region changes from having a penalty of $-1.6$ to $-107$. Overall, these visualizations show that the reward function automatically changes during training and provides a strong reward signal for different parts of the state space depending on the behavior of the policy.

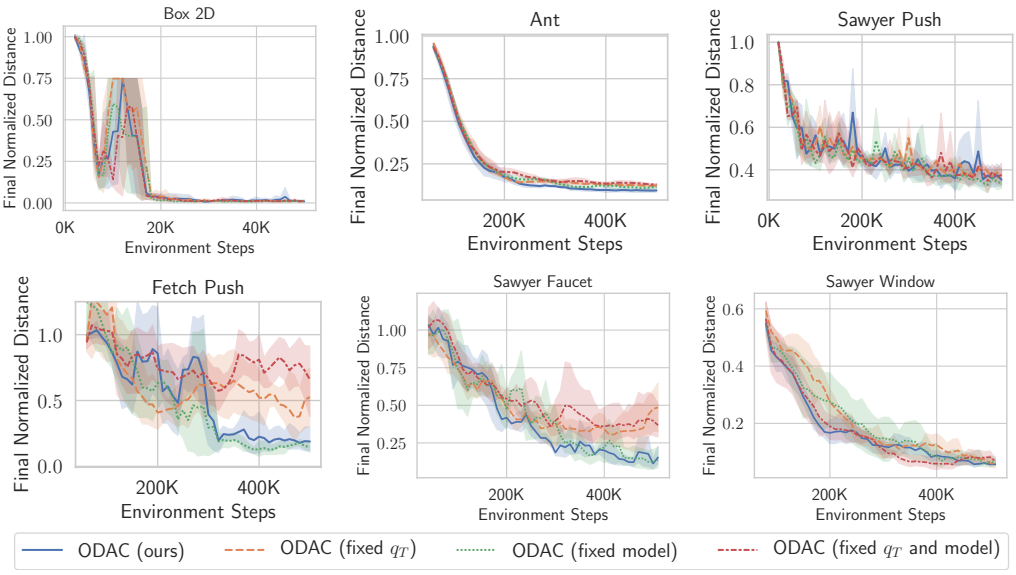



**Figure B.1:** Ablation results across all six environments. We see that using our derived $q_T$ equation is important for best performance across all six tasks and that ODAC is not sensitive to the quality of dynamics model.

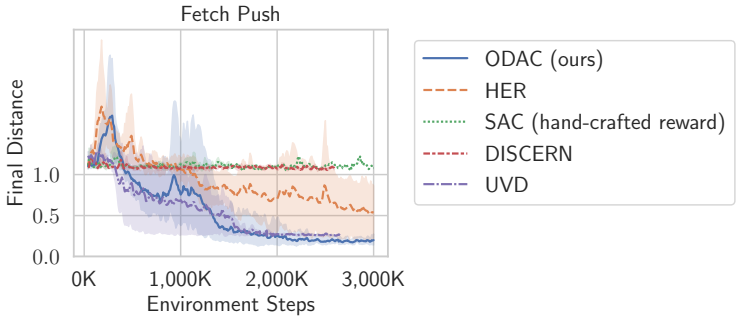



**Figure B.2:** Comparison of different methods when the desired outcome $\mathbf{g}$ is sampled uniformly from the set of possible outcomes during exploration. In this easier setting, we see that the UVD performance is similar to ODAC. These results suggest that UVD depends more heavily on sampling outcomes from the set of desired outcomes than ODAC.

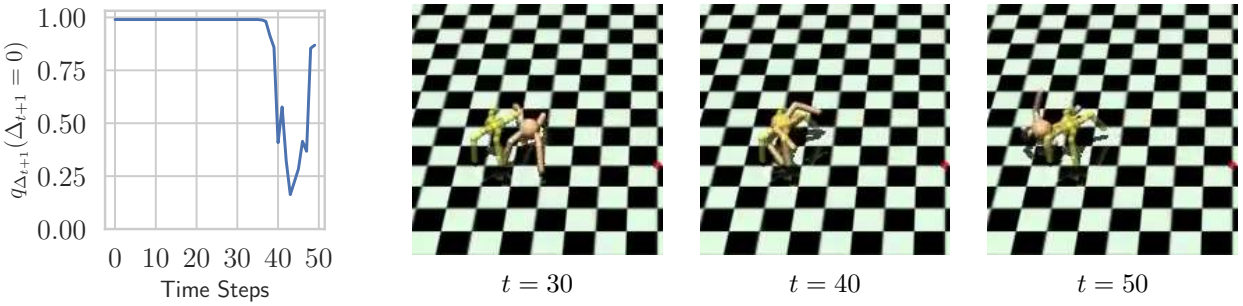
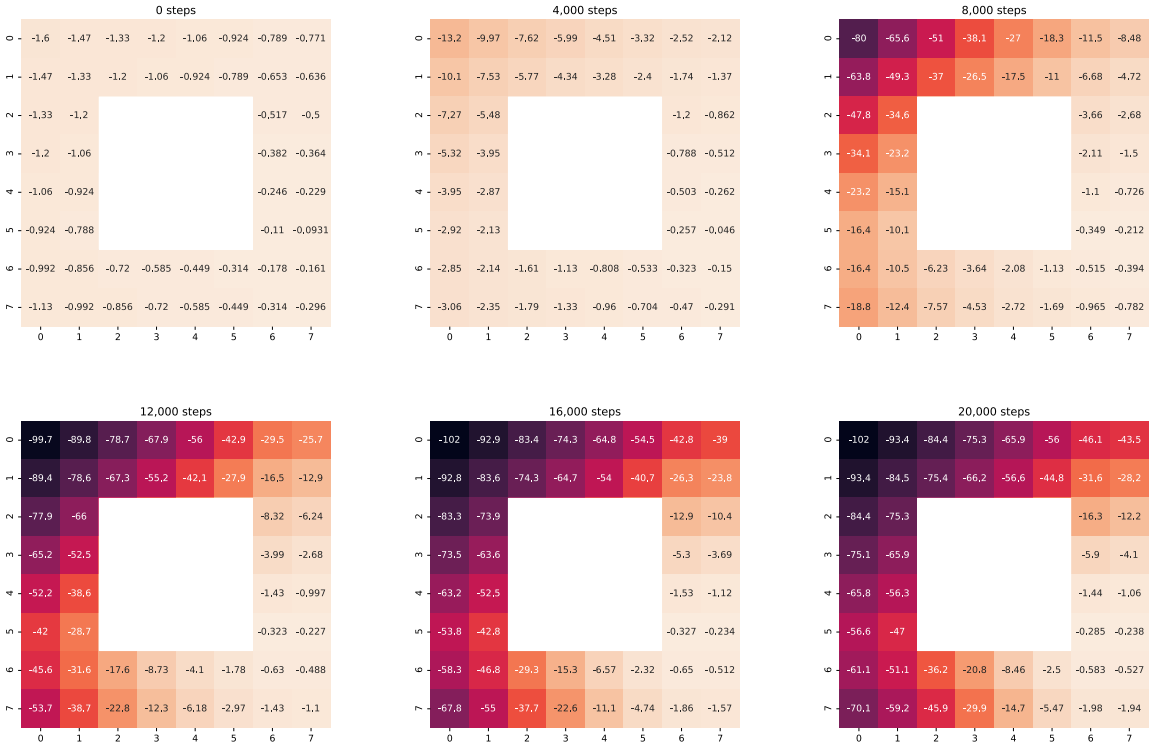


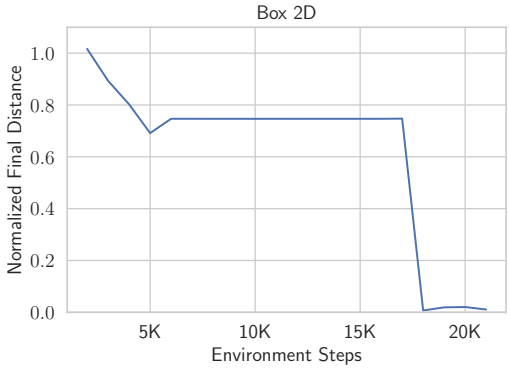

**Figure B.3:** The inferred $q_{\Delta_{t+1}}(\Delta_{t+1} = 0)$ versus time during an example trajectory in the Ant environment. As the ant robot falls over, $q_{\Delta_{t+1}}(\Delta_{t+1} = 0)$ drops in value. We see that the optimal posterior $q^{\star}_{\Delta_{t+1}}(\Delta_{t+1} = 0)$ given in Proposition 2 automatically assigns a high likelihood of terminating when this irrecoverable state is first reached, effectively acting as a dynamic discount factor.

| Environment | ODAC (Mean + Standard Error) | Dyna (Mean + Standard Error) |
|---|---|---|
| Box 2D | 0.74 (0.091) | 0.87 (0.058) |
| Ant | 33 (27) | 102 (0.83) |
| Sawyer Faucet | 14 (6.3) | 100 (5) |
| Fetch Push | 12 (3.7) | 96 (3.8) |
| Sawyer Push | 58 (8.7) | 96 (0.39) |
| Sawyer Window | 4.4 (1.5) | 116 (14) |

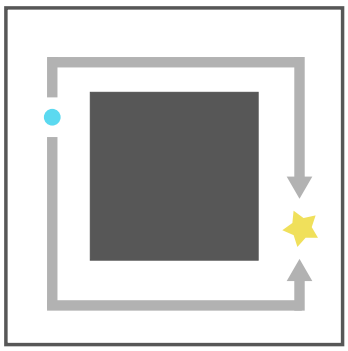
**Table B.1:** Normalized final distances (lower is better) across four random seeds, multiplied by a factor of 100.




**(a)** Reward Visualization




**(b)** Learning Curve associated with Reward Visualized



**(c)** Environment Visualization

**Figure B.4:** We visualize the rewards over the course of training for the Box 2D environment. The darkest color corresponds to a reward of $-100$ and brightest to 0. To visualize the reward, we discretize the continuous state space and evaluate $r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}; q_\Delta)$ for $\mathbf{a}_t = \vec{0}$ at different states. As shown in Figure B.4c, the desired outcome $\mathbf{g}$ is near the bottom right. After 4-8 thousand environment steps, the reward is more flat near $\mathbf{g}$, and only provides a reward gradient far from $\mathbf{g}$. After 20 thousand environment steps, the reward gradient is much larger again near the end, and the penalty for being in the top left corner has changed from $-1.6$ to $-107$.

## B.4 Experimental Details

### B.4.1 Environment

**Ant.** This Ant domain is based on the "Ant-V3" OpenAI Gym [20] environment, with three modifications: the gear ratio is reduced from 150 to 120, the contact force sensors are removed from the state, and there is no termination condition and the episode only terminates after a fixed amount of time. In this environment, the state space is 23 dimensional, consistent of the XYZ coordinate of the center of the torso, the orientation of the ant (in quaternion), and the angle and angular velocity of all 8 joints. The action space is 8-dimensional and corresponds to the torque to apply to each joint. The desired outcome consists of the desired XYZ, orientation, and joint angles at a position that is 5 meters down and to the right of the initial position. This desired pose is shown in Figure 2.4.

**Sawyer push.** In this environment, the state and goal space is 4 dimensional and the action space is 2 dimension. The state and goal consists of the XY end effector (EE) and the XY position of the puck. The object is on a 40cm x 50cm table and starts 20 cm in front of the hand. The goal puck position is fixed to 15 cm forward and 30 cm to the right of the initial hand position, while the goal hand position is 5cm behind and 20 cm to the right of the initial hand position. The action is the change in position in each XY direction, with a maximum change of 3 cm per direction at each time step. The episode horizon is 100.

**Box 2D.** In this environment, the state is a $4x4$ with a $2x2$ box in the middle. The policy is initialized to to $(-3.5, -2)$ and the desired outcome is $(3.5, 2)$. The action is the XY velocity of the agent, with wall collisions taken into account and maximum velocity of 0.2 in each direction. To make the environment stochastic, we add Gaussian noise to actions with mean zero and standard deviation that's 10% of the maximum action magnitude.

**Sawyer window and faucet.** In this environment, the state and goal space is 6 dimensional and the action space is 2 dimension. The state and goal consists of the XYZ end effector (EE) and the XYZ position of the window or faucet end endpoint. The hand is initialized away from the window and faucet. The EE goal XYZ position is set to the initial window or faucet position. The action is the change in position in each XYZ direction. For the window task, the goal positions is to close the window, and for the faucet task, the goal position is to rotate the faucet 90 degrees counter-clockwise from above.

### B.4.2 Algorithm

Pseudocode for the complete algorithm is shown in Algorithm 8.

**Algorithm 8** Outcome-Driven Actor Critic

**Require:** Policy $\pi_\theta$, $Q$-function $Q_\phi$, dynamics model $p_\psi$, replay buffer $\mathcal{R}$, and map from state to achieved goal $f$.
  **for** $n = 0, \dots N-1$ episodes **do**
    Sample initial state $\mathbf{s}_0$ from environment.
    Sample goal $\mathbf{g}$ from environment.
    **for** $t = 0, \dots, H-1$ steps **do**
      Get action $\mathbf{a}_t \sim \pi_\theta(\mathbf{s}_t, \mathbf{g})$.
      Get next state $\mathbf{s}_{t+1} \sim p(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)$.
      Store $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{g})$ into replay buffer $\mathcal{R}$.
      Sample transition $(\mathbf{s}, \mathbf{a}, \mathbf{s}', \mathbf{g}) \sim \mathcal{R}$.
      Compute reward $r = \log p_\psi(\mathbf{g} \mid \mathbf{s}, \mathbf{a}) - D_{\text{KL}}\left(q_\Delta(\cdot|\mathbf{s}_t, \mathbf{a}_t) \middle\| p(\Delta)\right)$.
      Compute $q(\Delta_t = 0|\mathbf{s}, \mathbf{a})$ using Equation (2.15).
      Update $Q_\phi$ using Equation (2.19) and data $(\mathbf{s}, \mathbf{a}, \mathbf{s}', \mathbf{g}, r)$.
      Update $\pi_\theta$ using Equation (2.20) and data $(\mathbf{s}, \mathbf{a}, \mathbf{g})$.
      Update $p_\psi$ using Equation (2.21) and data $(\mathbf{s}, \mathbf{a}, \mathbf{g})$.
    **end for**
    **for** $t = 0, ..., H-1$ steps **do**
      **for** $i = 0, ..., k-1$ steps **do**
        Sample future state $\mathbf{s}_{h_i}$, where $t < h_i \leq H-1$.
        Store $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, f(\mathbf{s}_{h_i}))$ into $\mathcal{R}$.
      **end for**
    **end for**
  **end for**

### B.4.3 Implementation

Table B.3 lists the hyper-parameters that were shared across the experiments. Table B.2 lists hyper-parameters specific to each environment. We give extra implementation details.

**Dynamics model.** For the Ant and Sawyer experiments, we train a neural network to output the mean and standard deviation of a Laplace distribution. This distribution is then used to model the distribution over the *difference* between the current state and the next state, which we found to be more reliable than predicting the next state. So, the overall distribution is given by a Laplace distribution with learned mean $\mu$ and fixed standard deviation $\sigma$ computed via

$$p_\psi = \text{Laplace}(\mu = g_\psi(\mathbf{s}, \mathbf{a}) + f(\mathbf{s}), \sigma = 0.00001)$$

where $g$ is the output of a network and $f$ is a function that maps a state into a goal.

For the 2D Navigation experiment, we use a Gaussian distribution. The dynamics neural network has hidden units of size $[64, 64]$ with a ReLU hidden activations. For the Ant and Sawyer experiments, there is no output activation. For the linear-Gaussiand and 2D

Navigation experiments, we have a tanh output, so that the mean and standard To bound the standard deviation outputted by the network, the standard-deviation tanh is multiplied by two with the standard deviation be between limited to between

**Reward normalization.** Because the different experiments have rewards of very different scale, we normalize the rewards by dividing by a running average of the maximum reward magnitude. Specifically, for every reward $r$ in the $i$th batch of data, we replace the reward with

$$\hat{r} = r/C_i$$

where we update the normalizing coefficient $C_i$ using each batch of reward $\{r_b\}_{b=1}^{B}$:

$$C_{i+1} \leftarrow (1-\lambda) \times C_i + \lambda \max_{b \in [1,\ldots,B]} |r_b|$$

and $C_i$ is initialized to 1. In our experiments, we use $\lambda = 0.001$.

**Target networks.** To train our $Q$-function, we use the technique from Fujimoto et al. [71] in which we train two separate Q-networks with target networks and take the minimum over two to compute the bootstrap value. The target networks are updated using a slow, moving average of the parameters after every batch of data:

$$\bar{\phi}_{i+1} = (1-\tau)\bar{\phi}_i + \times\phi_i.$$

In our experiments, we used $\tau = 0.001$.

**Automatic entropy tuning.** We use the same technique as in Haarnoja et al. [84] to weight the rewards against the policy entropy term. Specifically, we pre-multiply the entropy term in

$$\hat{V}(\mathbf{s}', \mathbf{g}) \approx Q_{\bar{\phi}}(\mathbf{s}', \mathbf{a}', \mathbf{g}) - \log \pi(\mathbf{a}'|\mathbf{s}'; \mathbf{g}),$$

by a parameter $\alpha$ that is updated to ensure that the policy entropy is above a minimum threshold. The parameter $\alpha$ is updated by taking a gradient step on the following function with each batch of data:

$$\mathcal{F}_\alpha(\alpha) = -\alpha \left(\log \pi(\mathbf{a} \mid \mathbf{s}, \mathbf{g}) + \mathcal{H}_{\text{target}}\right)$$

and where $\mathcal{H}_{\text{target}}$ is the target entropy of the policy. We follow the procedure in Haarnoja et al. [84] to choose $\mathcal{H}_{\text{target}}$ and choose $\mathcal{H}_{\text{target}} = -D_{\text{action}}$, where $D_{\text{action}}$ is the dimension of the action space.

| Environment | horizon | $Q$-function and policy hidden sizes |
|---|---|---|
| Box 2D | 100 | [64, 64] |
| Ant | 100 | [400, 300] |
| Fetch Push | 50 | [64, 64] |
| Sawyer Push | 100 | [400, 300] |
| Sawyer Window | 100 | [400, 300] |
| Sawyer Faucet | 100 | [400, 300] |

**Table B.2:** Environment specific hyper-parameters.

| **Hyper-parameter** | **Value** |
|---|---|
| # training batches per environment step | 1 |
| batch size | 256 |
| discount Factor | 0.99 |
| policy hidden activation | ReLU |
| $Q$-function hidden activation | ReLU |
| replay buffer size | 1 million |
| hindsight relabeling strategy | future |
| hindsight relabeling probability | 80% |
| target network update speed $\tau$ | 0.001 |
| reward scale update speed $\lambda$ | 0.001 |

**Table B.3:** General hyper-parameters used for all experiments.

**Exploration policy.** Because ODAC is an off-policy algorithm, we are free to use any exploration policy. It may be beneficial to add For the Ant and Sawyer tasks, we simply sample current policy. For the 2D Navigation task, at each time step, the policy takes a random action with probability 0.3 and repeats its

**Evaluation policy.** For evaluation, we use the mean of the learned policy for selecting actions.

# Appendix C

# Chapter 3 Appendix

## C.1 Section 3.1 Appendix

### C.1.1 Complete Ablative Results

#### C.1.1.1 Relabeling strategy ablation

In this experiment, we compare different goal resampling strategies for training the Q function. We consider: *Future*, relabeling the goal for a transition by sampling uniformly from future states in the trajectory as done in Andrychowicz et al. [3]; *VAE*, sampling goals from the VAE only; *RIG*, relabeling goals with probability 0.5 from the VAE and probability 0.5 using the future strategy; and *None*, no relabeling. Figure C.1 shows the effect of different relabeling strategies with our method.

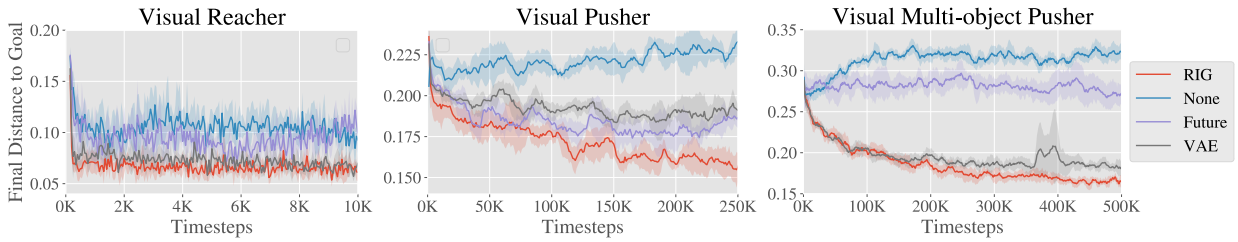



**Figure C.1:** Relabeling ablation simulated results, showing final distance to goal vs environment steps. RIG (red), which uses a mixture of VAE and future, consistently matches or outperforms the other methods.

#### C.1.1.2 Reward type ablation

In this experiment, we change only the reward function that we use to train the goal-conditioned valued function to show the effect of using the latent distance reward. We include the following methods for comparison: *Latent Distance*, which is the reward used

in RIG, i.e. $A = \mathbf{I}$ in Equation (3.1); *Log Probability*, which uses the Mahalanobis distance in Equation (3.1), where $A$ is the precision matrix of the encoder; and *Pixel MSE*, which computes mean-squared error (MSE) between state and goal in pixel space. To compute the pixel MSE for a sampled latent goal, we decode the goal latent using the VAE decoder, $p_\psi$, to generate the corresponding goal image. Figure C.2 shows the effect of different rewards with our method.

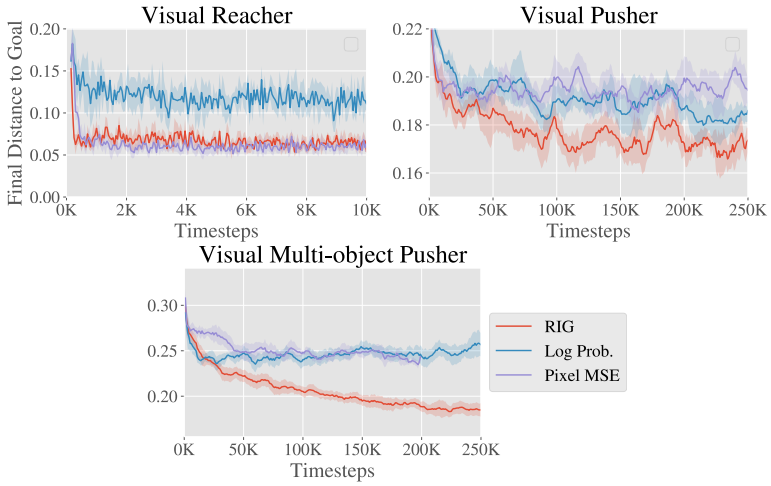



**Figure C.2:** Reward type ablation simulated results, showing final distance to goal vs environment steps. RIG (red), which uses latent distance for the reward, consistently matches or outperforms the other reward types.

### C.1.1.3 Online training ablation

Rather than pre-training the VAE on a set of images collected by a random policy, here we train the VAE in an online manner: the VAE is not trained when we initially collect data with our policy. After every 3000 environment steps, we train the VAE on all of the images observed by the policy. We show in Figure C.3 that this online training results in a good policy and is substantially better than leaving the VAE untrained. These results show that the representation learning can be done simultaneously as the reinforcement learning portion of RIG, eliminating the need to have a predefined set of images to train the VAE.

The Visual Pusher experiment for this ablation is performed on a slightly easier version of the Visual Pusher used for the main results. In particular, the goal space is reduced to be three quarters of its original size in the lateral dimension.

### C.1.1.4 Comparison to Hindsight Experience Replay

In this section, we study in isolation the effect of sampling goals from the goal space directly for $Q$-learning, as covered in Section 3.1.1.3. Like hindsight experience replay [3], in this

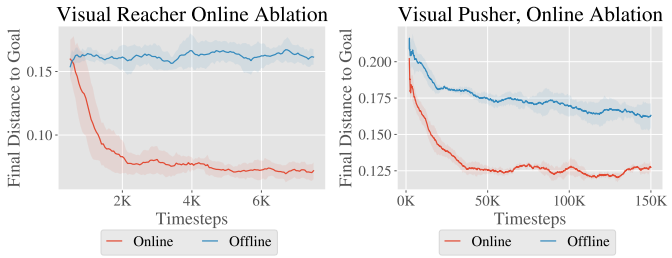



**Figure C.3:** Online vs offline VAE training ablation simulated results, showing final distance to goal vs environment steps. Given no pre-training phase, training the VAE online (red), outperforms no training of the VAE, and also performs well.

section we assume access to state information and the goal space, so we do not use a VAE.

To match the original work as closely as possible, this comparison was based off of the OpenAI baselines code [171] and we compare on the same Fetch robotics tasks. To minimize sample complexity and due to computational constraints, we use single threaded training with `rollout_batch_size=1, n_cycles=1, batch_size=256`. For testing, `n_test_rollouts=1` and the results are averaged over the last 100 test episodes. Number of updates per cycle corresponds to `n_batches`.

On the plots, "Future" indicates the future strategy as presented in Andrychowicz et al. [3] with $k = 4$. "Ours" indicates resampling goals with probability 0.5 from the "future" strategy with $k = 4$ and probability 0.5 uniformly from the environment goal space. Each method is shown with dense and sparse rewards.

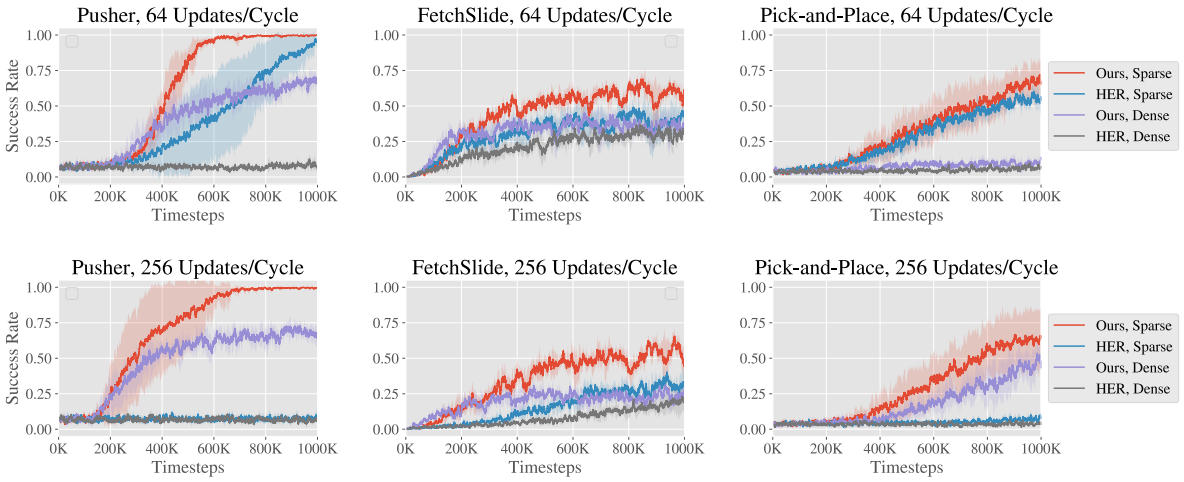



**Figure C.4:** Comparison between our relabeling strategy and HER. Each column shows a different task from the OpenAI Fetch robotics suite. The top row uses 64 gradient updates per training cycle and the bottom row uses 256 updates per cycle. Our relabeling strategy is significantly better for both sparse and dense rewards, and for higher number of updates per cycle.

Results are shown in Figure C.4. Our resampling strategy with sparse rewards consistently performs the best on the three tasks. Furthermore, it performs reasonably well with dense rewards, unlike HER alone which often fails with dense rewards. While the evaluation metric used here, success rate, is favorable to the sparse reward setting, learning with dense rewards is usually more sample efficient on most tasks and being able to do off-policy goal relabeling with dense rewards is important for RIG.

Finally, as the number of gradient updates per training cycle is increased, the performance of our strategy improves while HER does not improve and sometimes performs worse. As we apply reinforcement learning to real-world tasks, being able to reduce the required number of samples on hardware is one of the key bottlenecks. Increasing the number of gradient updates costs more compute but reduces the number of samples required to learn the tasks.

### C.1.2 Hyperparameters

Table C.1 lists the hyperparameters used for the experiments.

| Hyperparameter | Value |
|---|---|
| Mixture coefficient $\lambda$ | 0.5 |
| # training batches per time step | 4 |
| Exploration Policy | OU, $\theta = 0.15, \sigma = 0.3$ |
| $\beta$ for $\beta$-VAE | 5 |
| Critic Learning Rate | $10^{-3}$ |
| Critic Regularization | None |
| Actor Learning Rate | $10^{-3}$ |
| Actor Regularization | None |
| Optimizer | Adam |
| Target Update Rate ($\tau$) | $10^{-2}$ |
| Target Update Period | 2 time steps |
| Target Policy Noise | 0.2 |
| Target Policy Noise Clip | 0.5 |
| Batch Size | 128 |
| Discount Factor | 0.99 |
| Reward Scaling | $10^{-4}$ |
| Normalized Observations | False |
| Gradient Clipping | False |

**Table C.1:** Hyper-parameters used for all experiments.

### C.1.3 Environment Details

Below we provide a more detailed description of the simulated environments.

*Visual Reacher*: A MuJoCo environment with a 7-DoF Sawyer arm reaching goal positions. The arm is shown on the left of Figure 3.2 with two extra objects for the Visual Multi-Object Pusher environment (see below). The end-effector (EE) is constrained to a 2-dimensional rectangle parallel to a table. The action controls EE velocity within a maximum velocity. The underlying state is the EE position $e$, and the underlying goal is to reach a desired EE position, $g_e$.

*Visual Pusher*: A MuJoCo environment with a 7-DoF Sawyer arm and a small puck on a table that the arm must push to a target position. Control is the same as in Visual Reacher. The underlying state is the EE position, $e$ and puck position $p$. The underlying goal is for the EE to reach a desired position $g_e$ and the puck to reach a desired position $p$.

*Visual Multi-Object Pusher*: A copy of the Visual Pusher environment with two pucks. The underlying state is the EE position, $e$ and puck positions $p_1$ and $p_2$. The underlying goal is for the EE to reach desired position $g_e$ and the pucks to reach desired positions $g_1$ and $g_2$ in their respective halves of the workspace. Each puck and respective goal is initialized in half of the workspace.

Videos of our method in simulated and real-world environments can be found at `https://sites.google.com/site/visualrlwithimaginedgoals/`.

## C.2 Section 3.2 Appendix

### C.2.1 Proofs

The definitions of continuity and convergence for pseudo-metrics are similar to those for metrics, and we state them below.

A function $f : \mathcal{Q} \mapsto \mathcal{Q}$ is continuous with respect to a pseudo-metric $d$ if for any $p \in \mathcal{Q}$ and any scalar $\epsilon > 0$, there exists a $\delta$ such that for all $q \in \mathcal{Q}$,

$$d(p, q) < \delta \implies d(f(p), f(q)) < \epsilon.$$

An infinite sequence $p_1, p_2 \dots$ converges to a value $p$ with respect to a pseudo-metric $d$, which we write as

$$\lim_{t \to \infty} p_t \to p,$$

if

$$\lim_{t \to \infty} d(p_t, p) \to 0.$$

Note that if $f$ is continuous, then

$$\lim_{t \to \infty} d_{\mathcal{H}}(p_t, q) \to 0 \implies \lim_{t \to \infty} d_{\mathcal{H}}(f(p_t), f(q)) \to 0.$$

### C.2.1.1 Proof of Lemma 3.1

**Lemma 8.** *Let $\mathcal{S}$ be a compact set. Define the set of distributions $\mathcal{Q} = \{p : support(p) \subseteq \mathcal{S}\}$. Let $\mathcal{F} : \mathcal{Q} \mapsto \mathcal{Q}$ be continuous with respect to the pseudometric $d_{\mathcal{H}}(p, q) \doteq |\mathcal{H}(p) - \mathcal{H}(q)|$ and $\mathcal{H}(\mathcal{F}(p)) \geq \mathcal{H}(p)$ with equality if and only if $p$ is the uniform probability distribution on $\mathcal{S}$, denoted as $U_{\mathcal{S}}$. Define the sequence of distributions $P = (p_1, p_2, \dots)$ by starting with any $p_1 \in \mathcal{Q}$ and recursively defining $p_{t+1} = \mathcal{F}(p_t)$. The sequence $P$ converges to $U_{\mathcal{S}}$ with respect to $d_{\mathcal{H}}$. In other words, $\lim_{t\to 0} |\mathcal{H}(p_t) - \mathcal{H}(U_{\mathcal{S}})| \to 0$.*

*Proof.* The idea of the proof is to show that the distance (with respect to $d_{\mathcal{H}}$) between $p_t$ and $U_{\mathcal{S}}$ converges to a value. If this value is 0, then the proof is complete since $U_{\mathcal{S}}$ uniquely has zero distance to itself. Otherwise, we will show that this implies that $\mathcal{F}$ is not continuous, which a contradiction.

For shorthand, define $d_t$ to be the $d_{\mathcal{H}}$-distance to the uniform distribution, as in

$$d_t \doteq d_{\mathcal{H}}(p_t, U_{\mathcal{S}}).$$

First we prove that $d_t$ converges. Since the entropies of the sequence $(p_1, \dots)$ monotonically increase, we have that

$$d_1 \geq d_2 \geq \dots.$$

We also know that $d_t$ is lower bounded by 0, and so by the monotonic convergence theorem, we have that

$$\lim_{t\to\infty} d_t \to d^*.$$

for some value $d^* \geq 0$.

To prove the lemma, we want to show that $d^* = 0$. Suppose, for contradiction, that $d^* \neq 0$. Then consider any distribution, $q^*$, such that $d_{\mathcal{H}}(q^*, U_{\mathcal{S}}) = d^*$. Such a distribution always exists since we can continuously interpolate entropy values between $\mathcal{H}(p_1)$ and $\mathcal{H}(U_{\mathcal{S}})$ with a mixture distribution. Note that $q^* \neq U_{\mathcal{S}}$ since $d_{\mathcal{H}}(U_{\mathcal{S}}, U_{\mathcal{S}}) = 0$. Since $\lim_{t\to\infty} d_t \to d^*$, we have that

$$\lim_{t\to\infty} d_{\mathcal{H}}(p_t, q^*) \to 0, \tag{C.1}$$

and so

$$\lim_{t\to\infty} p_t \to q^*.$$

Because the function $\mathcal{F}$ is continuous with respect to $d_{\mathcal{H}}$, Equation C.1 implies that

$$\lim_{t\to\infty} d_{\mathcal{H}}(\mathcal{F}(p_t), \mathcal{F}(q^*)) \to 0.$$

However, since $\mathcal{F}(p_t) = p_{t+1}$ we can equivalently write the above equation as

$$\lim_{t\to\infty} d_{\mathcal{H}}(p_{t+1}, \mathcal{F}(q^*)) \to 0.$$

which, through a change of index variables, implies that

$$\lim_{t\to\infty} p_t \to \mathcal{F}(q^*)$$

Since $q^*$ is not the uniform distribution, we have that $\mathcal{H}(\mathcal{F}(q^*)) > \mathcal{H}(q^*)$, which implies that $\mathcal{F}(q^*)$ and $q^*$ are unique distributions. So, $p_t$ converges to two distinct values, $q^*$ and $\mathcal{F}(q^*)$, which is a contradiction. Thus, it must be the case that $d^* = 0$, completing the proof. □

#### C.2.1.2 Proof of Lemma 3.2

**Lemma 9.** *Given two distribution $p(x)$ and $q(x)$ where $p \ll q$ and*

$$0 < \mathrm{Cov}_p[\log p(X), \log q(X)] \tag{C.2}$$

*define the distribution $p_\alpha$ as*

$$p_\alpha(x) = \frac{1}{Z_\alpha} p(x) q(x)^\alpha$$

*where $\alpha \in \mathbb{R}$ and $Z_\alpha$ is the normalizing factor. Let $\mathcal{H}_\alpha(\alpha)$ be the entropy of $p_\alpha$. Then there exists a constant $a > 0$ such that for all $\alpha \in [-a, 0)$,*

$$\mathcal{H}_\alpha(\alpha) > \mathcal{H}_\alpha(0) = \mathcal{H}(p). \tag{C.3}$$

*Proof.* Observe that $\{p_\alpha : \alpha \in [-1, 0]\}$ is a one-dimensional exponential family

$$p_\alpha(x) = e^{\alpha T(x) - A(\alpha) + k(x)}$$

with log carrier density $k(x) = \log p(x)$, natural parameter $\alpha$, sufficient statistic $T(x) = \log q(x)$, and log-normalizer $A(\alpha) = \int_{\mathcal{X}} e^{\alpha T(x) + k(x)} dx$. As shown in [160], the entropy of a distribution from a one-dimensional exponential family with parameter $\alpha$ is given by:

$$\mathcal{H}_\alpha(\alpha) \doteq \mathcal{H}(p_\alpha) = A(\alpha) - \alpha A'(\alpha) - \mathbb{E}_{p_\alpha}[k(X)]$$

The derivative with respect to $\alpha$ is then

$$\begin{aligned}
\frac{d}{d\alpha}\mathcal{H}_\alpha(\alpha) &= -\alpha A''(\alpha) - \frac{d}{d\alpha}\mathbb{E}_{p_\alpha}[k(x)] \\
&= -\alpha A''(\alpha) - \mathbb{E}_\alpha[k(x)(T(x) - A'(\alpha)] \\
&= -\alpha \mathrm{Var}_{p_\alpha}[T(x)] - \mathrm{Cov}_{p_\alpha}[k(x), T(x)]
\end{aligned}$$

where we use the fact that the $n$th derivative of $A(\alpha)$ give the $n$ central moment, i.e. $A'(\alpha) = \mathbb{E}_{p_\alpha}[T(x)]$ and $A''(\alpha) = \mathrm{Var}_{p_\alpha}[T(x)]$. The derivative of $\alpha = 0$ is

$$\begin{aligned}\frac{d}{d\alpha}\mathcal{H}_\alpha(0) &= -\mathrm{Cov}_{p_0}[k(x), T(x)] \\ &= -\mathrm{Cov}_p[\log p(x), \log q(x)]\end{aligned}$$

which is negative by assumption. Because the derivative at $\alpha = 0$ is negative, then there exists a constant $a > 0$ such that for all $\alpha \in [-a, 0]$, $\mathcal{H}_\alpha(\alpha) > \mathcal{H}_\alpha(0) = \mathcal{H}(p)$. □

The paper applies Lemma 9 to the case where $q = q_\phi^G$ and $p = p_\phi^S$. When we take $N \to \infty$, we have that $p_{\text{skewed}}$ corresponds to $p_\alpha$ above.

### C.2.1.3 Simple Case Proof

We prove the convergence directly for the (even more) simplified case when $p_\theta = p(\mathbf{S} \mid q_{\phi_t}^G)$ using a similar technique:

**Lemma 10.** *Assume the set $\mathcal{S}$ has finite volume so that its uniform distribution $U_\mathcal{S}$ is well defined and has finite entropy. Given any distribution $p(\mathbf{s})$ whose support is $\mathcal{S}$, recursively define $p_t$ with $p_1 = p$ and*

$$p_{t+1}(\mathbf{s}) = \frac{1}{Z_\alpha^t} p_t(\mathbf{s})^\alpha, \quad \forall \mathbf{s} \in \mathcal{S}$$

*where $Z_\alpha^t$ is the normalizing constant and $\alpha \in [0, 1)$.*

*The sequence $(p_1, p_2, \dots)$ converges to $U_\mathcal{S}$, the uniform distribution $\mathcal{S}$.*

*Proof.* If $\alpha = 0$, then $p_2$ (and all subsequent distributions) will clearly be the uniform distribution. We now study the case where $\alpha \in (0, 1)$.

At each iteration $t$, define the one-dimensional exponential family $\{p_\theta^t : \theta \in [0, 1]\}$ where $p_\theta^t$ is

$$p_\theta^t(\mathbf{s}) = e^{\theta T(\mathbf{s}) - A(\theta) + k(\mathbf{s})}$$

with log carrier density $k(\mathbf{s}) = 0$, natural parameter $\theta$, sufficient statistic $T(\mathbf{s}) = \log p_t(\mathbf{s})$, and log-normalizer $A(\theta) = \int_\mathcal{S} e^{\theta T(\mathbf{s})} d\mathbf{s}$. As shown in [160], the entropy of a distribution from a one-dimensional exponential family with parameter $\theta$ is given by:

$$\mathcal{H}_\theta^t(\theta) \doteq \mathcal{H}(p_\theta^t) = A(\theta) - \theta A'(\theta)$$

The derivative with respect to $\theta$ is then

$$\begin{aligned}\frac{d}{d\theta} d\mathcal{H}_\theta^t(\theta) &= -\theta A''(\theta) \\ &= -\theta \mathrm{Var}_{\mathbf{s} \sim p_\theta^t}[T(\mathbf{s})] \\ &= -\theta \mathrm{Var}_{\mathbf{s} \sim p_\theta^t}[\log p_t(\mathbf{s})] \\ &\leq 0\end{aligned} \tag{C.4}$$

where we use the fact that the $n$th derivative of $A(\theta)$ is the $n$ central moment, i.e. $A''(\theta) = \text{Var}_{\mathbf{s}\sim p_\theta^t}[T(\mathbf{s})]$. Since variance is always non-negative, this means the entropy is monotonically decreasing with $\theta$. Note that $p_{t+1}$ is a member of this exponential family, with parameter $\theta = \alpha \in (0, 1)$. So

$$\mathcal{H}(p_{t+1}) = \mathcal{H}_\theta^t(\alpha) \geq \mathcal{H}_\theta^t(1) = \mathcal{H}(p_t)$$

which implies

$$\mathcal{H}(p_1) \leq \mathcal{H}(p_2) \leq \dots.$$

This monotonically increasing sequence is upper bounded by the entropy of the uniform distribution, and so this sequence must converge.

The sequence can only converge if $\frac{d}{d\theta}\mathcal{H}_\theta^t(\theta)$ converges to zero. However, because $\alpha$ is bounded away from 0, Equation (C.4) states that this can only happen if

$$\text{Var}_{\mathbf{s}\sim p_\theta^t}[\log p_t(\mathbf{s})] \to 0. \tag{C.5}$$

Because $p_t$ has full support, then so does $p_\theta^t$. Thus, Equation (C.5) is only true if $\log p_t(\mathbf{s})$ converges to a constant, i.e. $p_t$ converges to the uniform distribution. □

## C.2.2 Additional Experiments

### C.2.2.1 Sensitivity Analysis

**Sensitivity to RL Algorithm** In our experiments, we combined Skew-Fit with soft actor critic (SAC) [84]. We conduct a set of experiments to test whether Skew-Fit may be used with other RL algorithms for training the goal-conditioned policy. To that end, we replaced SAC with twin delayed deep deterministic policy gradient (TD3) [71] and ran the same Skew-Fit experiments on Visual Door, Visual Pusher, and Visual Pickup. In Figure C.5, we see that Skew-Fit performs consistently well with both SAC and TD3, demonstrating that Skew-Fit is beneficial across multiple RL algorithms.

**Sensitivity to $\alpha$ Hyperparameter** We study the sensitivity of the $\alpha$ hyperparameter by testing values of $\alpha \in [-1, -0.75, -0.5, -0.25, 0]$ on the Visual Door and Visual Pusher task. The results are included in Figure C.6 and shows that our method is robust to different parameters of $\alpha$, particularly for the more challenging Visual Pusher task. Also, the method consistently outperform $\alpha = 0$, which is equivalent to sampling uniformly from the replay buffer.

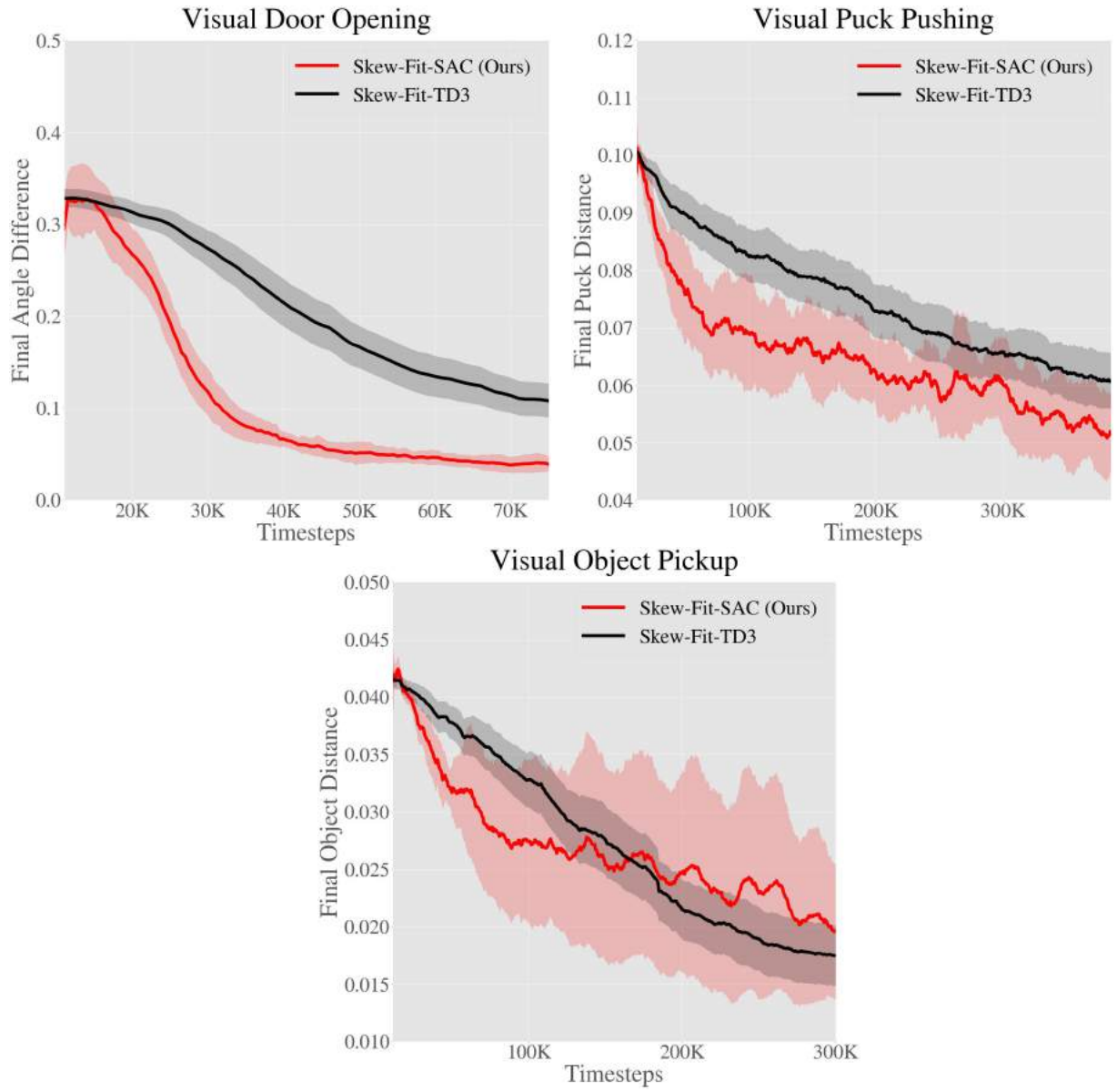



**Figure C.5:** We compare using SAC [84] and TD3 [71] as the underlying RL algorithm on Visual Door, Visual Pusher and Visual Pickup. We see that Skew-Fit works consistently well with both SAC and TD3, demonstrating that Skew-Fit may be used with various RL algorithms. For the experiments presented in subsection 3.2.4, we used SAC.

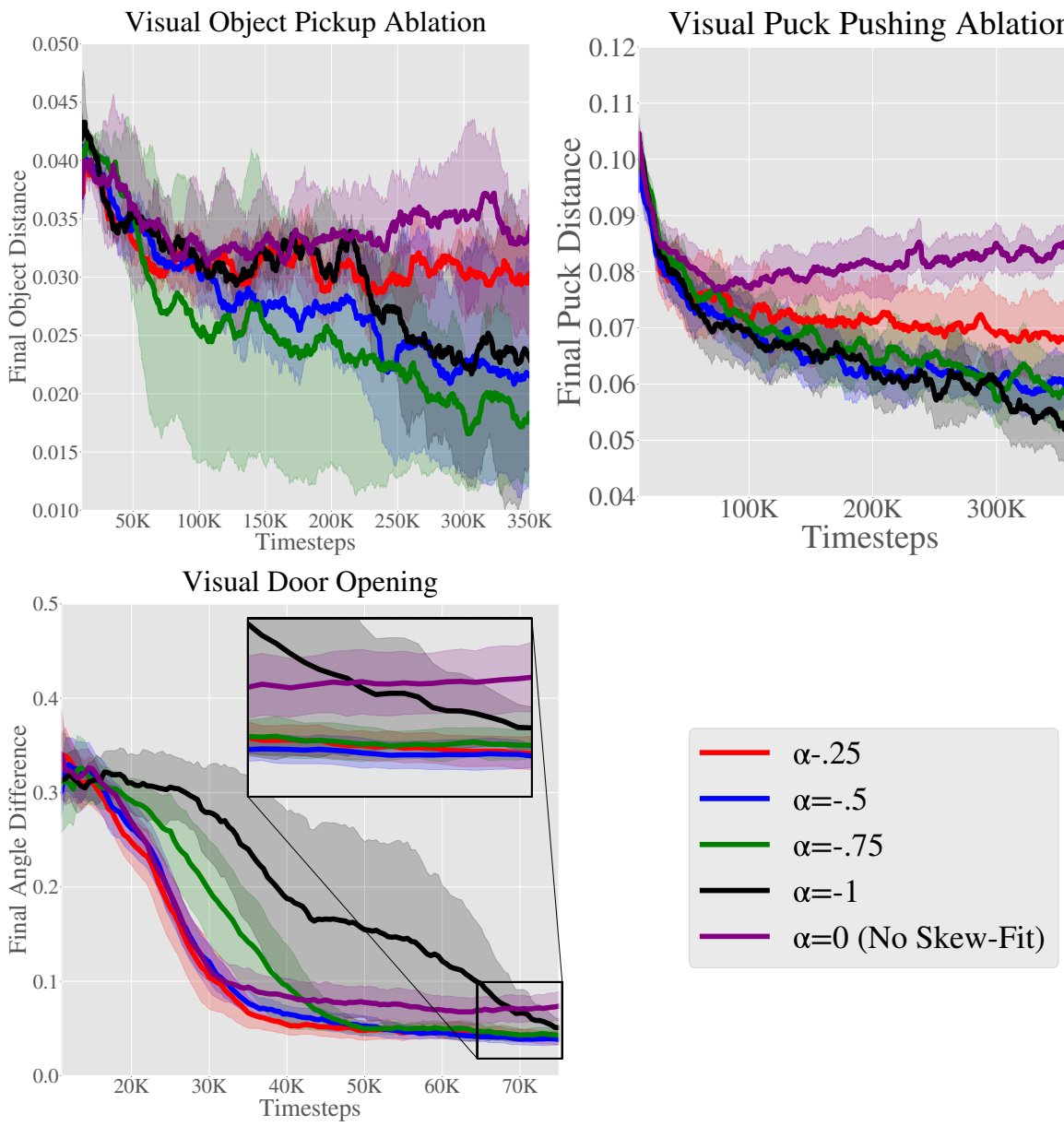



**Figure C.6:** We sweep different values of $\alpha$ on Visual Door, Visual Pusher and Visual Pickup. Skew-Fit helps the final performance on the Visual Door task, and outperforms No Skew-Fit ($\alpha = 0$) as seen in the zoomed in version of the plot. In the more challenging Visual Pusher task, we see that Skew-Fit consistently helps and halves the final distance. Similarly, we observe that Skew-Fit consistently outperforms No Skew-fit on Visual Pickup. Note that alpha=-1 is not always the optimal setting for each environment, but outperforms $\alpha = 0$ in each case in terms of final performance.

| Method | NLL |
|---|---|
| MLE on uniform (oracle) | 20175.4 |
| Skew-Fit on unbalanced | 20175.9 |
| MLE on unbalanced | 20178.03 |

**Table C.2:** Despite training on a unbalanced Visual Door dataset (see Figure 7 of paper), the negative log-likelihood (NLL) of Skew-Fit evaluated on a uniform dataset matches that of a VAE trained on a uniform dataset.

### C.2.2.2 Variance Ablation

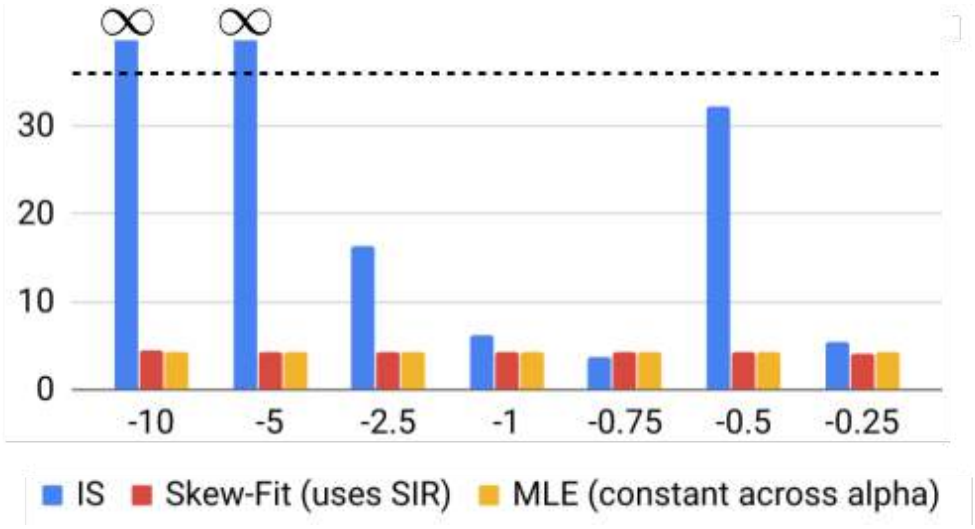



**Figure C.7:** Gradient variance averaged across parameters in last epoch of training VAEs. Values of $\alpha$ less than $-1$ are numerically unstable for importance sampling (IS), but not for Skew-Fit.

We measure the gradient variance of training a VAE on an unbalanced Visual Door image dataset with Skew-Fit vs Skew-Fit with importance sampling (IS) vs no Skew-Fit (labeled MLE). We construct the imbalanced dataset by rolling out a random policy in the environment and collecting the visual observations. Most of the images contained the door in a closed position; in a few, the door was opened. In Figure C.7, we see that the gradient variance for Skew-Fit with IS is catastrophically large for large values of $\alpha$. In contrast, for Skew-Fit with SIR, which is what we use in practice, the variance is relatively similar to that of MLE. Additionally we trained three VAE's, one with MLE on a uniform dataset of valid door opening images, one with Skew-Fit on the unbalanced dataset from above, and one with MLE on the same unbalanced dataset. As expected, the VAE that has access to the uniform dataset gets the lowest negative log likelihood score. This is the oracle method, since in practice we would only have access to imbalanced data. As shown in Table C.2, Skew-Fit considerably outperforms MLE, getting a much closer to oracle log likelihood score.

#### C.2.2.3 Goal and Performance Visualization

We visualize the goals sampled from Skew-Fit as well as those sampled when using the prior method, RIG [155]. As shown in Figure C.8 and Figure C.9, the generative model $q_\phi^G$ results in much more diverse samples when trained with Skew-Fit. We we see in Figure C.10, this results in a policy that more consistently reaches the goal image.

### C.2.3 Implementation Details

#### C.2.3.1 Likelihood Estimation using $\beta$-VAE

We estimate the density under the VAE by using a sample-wise approximation to the marginal over $x$ estimated using importance sampling:

$$\begin{aligned} q_{\phi_t}^G(x) &= \mathbb{E}_{z\sim q_{\theta_t}(z|x)}\left[\frac{p(z)}{q_{\theta_t}(z|x)} p_{\psi_t}(x \mid z)\right] \\ &\approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{p(z)}{q_{\theta_t}(z|x)} p_{\psi_t}(x \mid z)\right]. \end{aligned}$$

where $q_\theta$ is the encoder, $p_\psi$ is the decoder, and $p(z)$ is the prior, which in this case is unit Gaussian. We found that sampling $N = 10$ latents for estimating the density worked well in practice.

#### C.2.3.2 Oracle 2D Navigation Experiments

We initialize the VAE to the bottom left corner of the environment for *Four Rooms*. Both the encoder and decoder have 2 hidden layers with [400, 300] units, ReLU hidden activations, and no output activations. The VAE has a latent dimension of 8 and a Gaussian decoder trained with a fixed variance, batch size of 256, and 1000 batches at each iteration. The VAE is trained on the exploration data buffer every 1000 rollouts.

#### C.2.3.3 Implementation of SAC and Prior Work

For all experiments, we trained the goal-conditioned policy using soft actor critic (SAC) [84]. To make the method goal-conditioned, we concatenate the target XY-goal to the state vector. During training, we retroactively relabel the goals [104, 3] by sampling from the goal distribution with probabilty 0.5. Note that the original RIG [155] paper used TD3 [71], which we also replaced with SAC in our implementation of RIG. We found that maximum entropy policies in general improved the performance of RIG, and that we did not need to add noise on top of the stochastic policy's noise. In the prior RIG method, the VAE was pre-trained on a uniform sampling of images from the state space of each environment. In order to ensure a fair comparison to Skew-Fit, we forego pre-training and instead train the VAE alongside RL, using the variant described in the RIG paper. For our RL network architectures and training

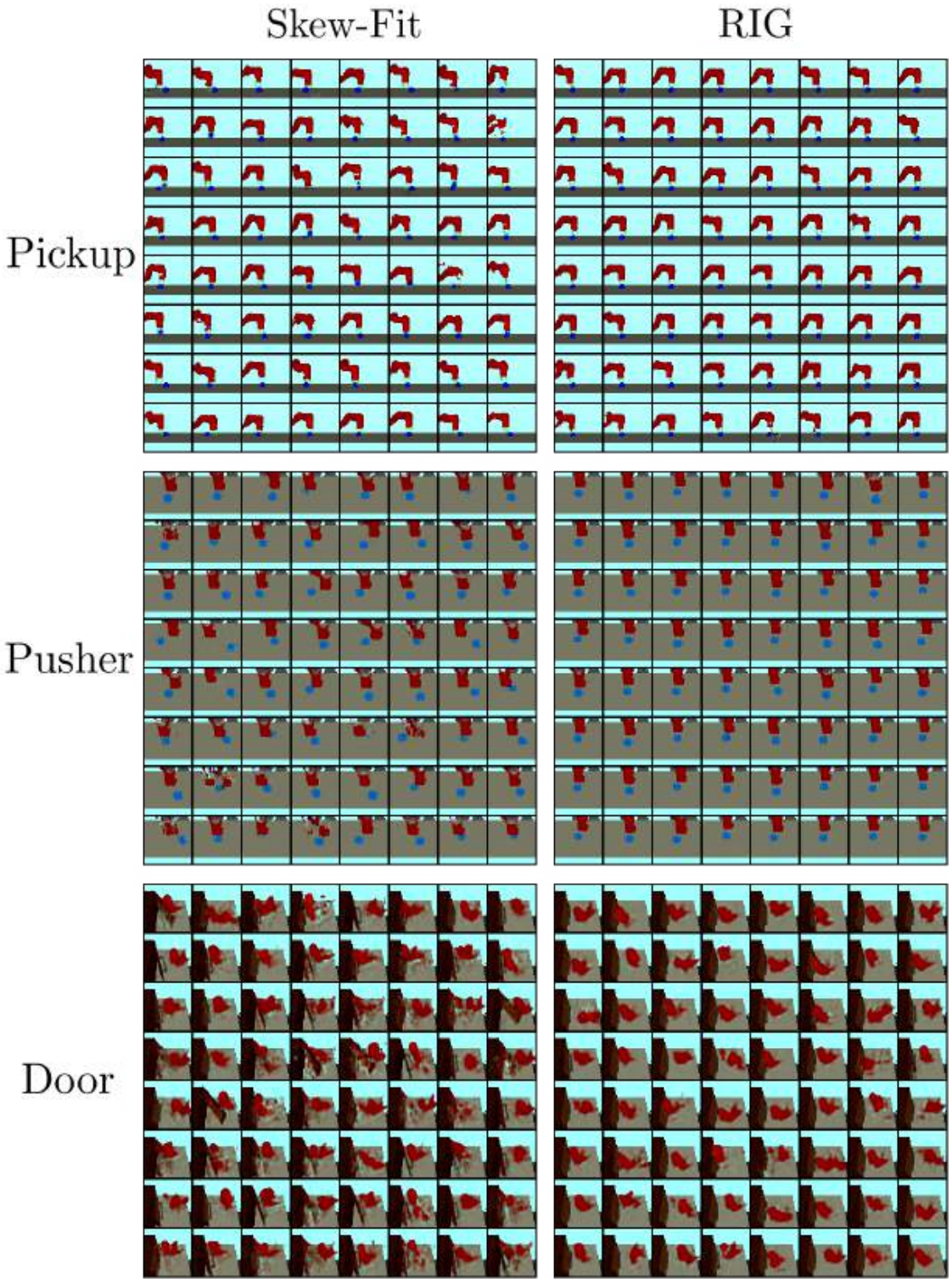



**Figure C.8:** Proposed goals from the VAE for RIG and with Skew-Fit on the *Visual Pickup*, *Visual Pusher*, and *Visual Door* environments. Standard RIG produces goals where the door is closed and the object and puck is in the same position, while RIG + Skew-Fit proposes goals with varied puck positions, occasional object goals in the air, and both open and closed door angles.

**Figure C.9:** Proposed goals from the VAE for RIG (left) and with RIG + Skew-Fit (right) on the *Real World Visual Door* environment. Standard RIG produces goals where the door is closed while RIG + Skew-Fit proposes goals with both open and closed door angles.

scheme, we use fully connected networks for the policy, $Q$-function and value networks with two hidden layers of size 400 and 300 each. We also delay training any of these networks for 10000 time steps in order to collect sufficient data for the replay buffer as well as to ensure the latent space of the VAE is relatively stable (since we continuously train the VAE concurrently with RL training). As in RIG, we train a goal-conditioned value functions [191] using hindsight experience replay [3], relabelling 50% of exploration goals as goals sampled from the VAE prior $\mathcal{N}(0, 1)$ and 30% from future goals in the trajectory.

For our implementation of [91], we trained the policies with the reward

$$r(s) = r_{\text{Skew-Fit}}(s) + \lambda \cdot r_{\text{Hazan et al.}}(s)$$

For $r_{\text{Hazan et al.}}$, we use the reward described in Section 5.2 of Hazan et al. [91], which requires an estimated likelihood of the state. To compute these likelihood, we use the same method as in Skew-Fit (see subsubsection C.2.3.1). With 3 seeds each, we tuned $\lambda$ across values $[100, 10, 1, 0.1, 0.01, 0.001]$ for the door task, but all values performed poorly. For the pushing and picking tasks, we tested values across $[1, 0.1, 0.01, 0.001, 0.0001]$ and found that 0.1 and 0.01 performed best for each task, respectively.

#### C.2.3.4 RIG with Skew-Fit Summary

9 provides detailed pseudo-code for how we combined our method with RIG. Steps that were removed from the base RIG algorithm are highlighted in blue and steps that were added are highlighted in red. The main differences between the two are (1) not needing to pre-train the $\beta$-VAE, (2) sampling exploration goals from the buffer using $p_{\text{skewed}}$ instead of the VAE prior, (3) relabeling with replay buffer goals sampled using $p_{\text{skewed}}$ instead of from the VAE prior, and (4) training the VAE on replay buffer data data sampled using $p_{\text{skewed}}$ instead of uniformly.

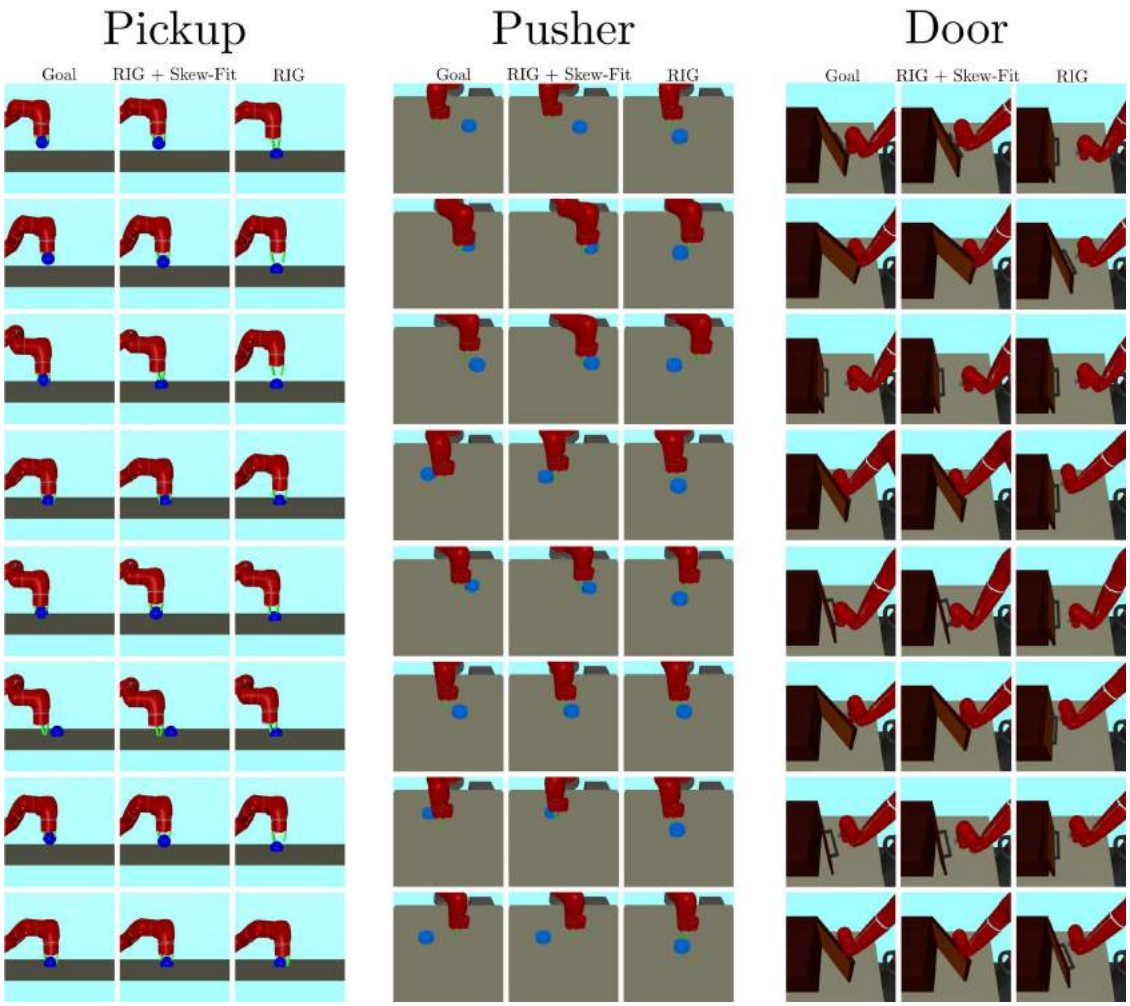



**Figure C.10:** Example reached goals by Skew-Fit and RIG. The first column of each environment section specifies the target goal while the second and third columns show reached goals by Skew-Fit and RIG. Both methods learn how to reach goals close to the initial position, but only Skew-Fit learns to reach the more difficult goals.

#### C.2.3.5 Vision-Based Continuous Control Experiments

In our experiments, we use an image size of 48x48. For our VAE architecture, we use a modified version of the architecture used in the original RIG paper [155]. Our VAE has three convolutional layers with kernel sizes: 5x5, 3x3, and 3x3, number of output filters: 16, 32, and 64 and strides: 3, 2, and 2. We then have a fully connected layer with the latent dimension number of units, and then reverse the architecture with de-convolution layers. We vary the latent dimension of the VAE, the $\beta$ term of the VAE and the $\alpha$ term for Skew-Fit based on the environment. Additionally, we vary the training schedule of the VAE based on the environment. See the table at the end of the appendix for more details. Our VAE has a Gaussian decoder with identity variance, meaning that we train the decoder with a mean-squared error loss.

When training the VAE alongside RL, we found the following three schedules to be effective for different environments:

1. For first $5K$ steps: Train VAE using standard MLE training every 500 time steps for 1000 batches. After that, train VAE using Skew-Fit every 500 time steps for 200 batches.
2. For first $5K$ steps: Train VAE using standard MLE training every 500 time steps for 1000 batches. For the next $45K$ steps, train VAE using Skew-Fit every 500 steps for 200 batches. After that, train VAE using Skew-Fit every 1000 time steps for 200 batches.
3. For first $40K$ steps: Train VAE using standard MLE training every 4000 time steps for 1000 batches. Afterwards, train VAE using Skew-Fit every 4000 time steps for 200 batches.

We found that initially training the VAE without Skew-Fit improved the stability of the algorithm. This is due to the fact that density estimates under the VAE are constantly changing and inaccurate during the early phases of training. Therefore, it made little sense to use those estimates to prioritize goals early on in training. Instead, we simply train using MLE training for the first $5K$ timesteps, and after that we perform Skew-Fit according to the VAE schedules above. Table C.3 lists the hyper-parameters that were shared across the continuous control experiments. Table C.4 lists hyper-parameters specific to each environment. Additionally, subsubsection C.2.3.4 discusses the combined RIG + Skew-Fit algorithm.

### C.2.4 Environment Details

*Four Rooms*: A 20 x 20 2D pointmass environment in the shape of four rooms [201]. The observation is the 2D position of the agent, and the agent must specify a target 2D position as the action. The dynamics of the environment are the following: first, the agent is teleported to the target position, specified by the action. Then a Gaussian change in position with mean 0 and standard deviation 0.0605 is applied[1]. If the action would result in the agent moving

[1] In the main paper, we rounded this to 0.06, but this difference does not matter.

| Hyper-parameter | Value | Comments |
|---|---|---|
| # training batches per time step | 2 | Marginal improvements after 2 |
| Exploration Noise | None (SAC policy is stochastic) | Did not tune |
| RL Batch Size | 1024 | smaller batch sizes work as well |
| VAE Batch Size | 64 | Did not tune |
| Discount Factor | 0.99 | Did not tune |
| Reward Scaling | 1 | Did not tune |
| Policy Hidden Sizes | $[400, 300]$ | Did not tune |
| Policy Hidden Activation | ReLU | Did not tune |
| Q-Function Hidden Sizes | $[400, 300]$ | Did not tune |
| Q-Function Hidden Activation | ReLU | Did not tune |
| Replay Buffer Size | 100000 | Did not tune |
| Number of Latents for Estimating Density ($N$) | 10 | Marginal improvements beyond 10 |

**Table C.3:** General hyper-parameters used for all *visual* experiments.

| Hyper-parameter | Pusher | Door | Pickup | Real World Door |
|---|---|---|---|---|
| Path Length | 50 | 100 | 50 | 100 |
| $\beta$ for $\beta$-VAE | 20 | 20 | 30 | 60 |
| Latent Dimension Size | 4 | 16 | 16 | 16 |
| $\alpha$ for Skew-Fit | $-1$ | $-1/2$ | $-1$ | $-1/2$ |
| VAE Training Schedule | 2 | 1 | 2 | 1 |
| Sample Goals From | $q_\phi^G$ | $p_{\text{skewed}}$ | $p_{\text{skewed}}$ | $p_{\text{skewed}}$ |

**Table C.4:** Environment specific hyper-parameters for the *visual* experiments

| Hyper-parameter | Value |
|---|---|
| # training batches per time step | .25 |
| Exploration Noise | None (SAC policy is stochastic) |
| RL Batch Size | 512 |
| VAE Batch Size | 64 |
| Discount Factor | $\frac{299}{300}$ |
| Reward Scaling | 10 |
| Path length | 300 |
| Policy Hidden Sizes | $[400, 300]$ |
| Policy Hidden Activation | ReLU |
| Q-Function Hidden Sizes | $[400, 300]$ |
| Q-Function Hidden Activation | ReLU |
| Replay Buffer Size | 1000000 |
| Number of Latents for Estimating Density ($N$) | 10 |
| $\beta$ for $\beta$-VAE | 10 |
| Latent Dimension Size | 2 |
| $\alpha$ for Skew-Fit | $-2.5$ |
| VAE Training Schedule | 3 |
| Sample Goals From | $p_{\text{skewed}}$ |

**Table C.5:** Hyper-parameters used for the *ant* experiment.

**Algorithm 9** RIG and RIG + Skew-Fit. Blue text denotes RIG specific steps and red text denotes RIG + Skew-Fit specific steps

**Require:** $\beta$-VAE mean encoder $q_\phi$, $\beta$-VAE decoder $p_\psi$, policy $\pi_\theta$, goal-conditioned value function $Q_w$, skew parameter $\alpha$, VAE Training Schedule.

Collect $\mathcal{D} = \{s^{(i)}\}$ using random initial policy.
Train $\beta$-VAE on data uniformly sampled from $\mathcal{D}$.
Fit prior $p(z)$ to latent encodings $\{\mu_\phi(s^{(i)})\}$.
**for** $n = 0, ..., N-1$ episodes **do**
    Sample latent goal from prior $z_g \sim p(z)$.
    Sample state $s_g \sim p_{\text{skewed}_n}$ and encode $z_g = q_\phi(s_g)$ if $\mathcal{R}$ is nonempty. Otherwise sample $z_g \sim p(z)$
    Sample initial state $s_0$ from the environment.
    **for** $t = 0, ..., H-1$ steps **do**
        Get action $a_t \sim \pi_\theta(q_\phi(s_t), z_g)$.
        Get next state $s_{t+1} \sim p(\cdot \mid s_t, a_t)$.
        Store $(s_t, a_t, s_{t+1}, z_g)$ into replay buffer $\mathcal{R}$.
        Sample transition $(s, a, s', z_g) \sim \mathcal{R}$.
        Encode $z = q_\phi(s), z' = q_\phi(s')$.
        (Probability 0.5) replace $z_g$ with $z'_g \sim p(z)$.
        (Probability 0.5) replace $z_g$ with $q_\phi(s'')$ where $s'' \sim p_{\text{skewed}_n}$.
        Compute new reward $r = -||z' - z_g||$.
        Minimize Bellman Error using $(z, a, z', z_g, r)$.
    **end for**
    **for** $t = 0, ..., H-1$ steps **do**
        **for** $i = 0, ..., k-1$ steps **do**
            Sample future state $s_{h_i}$, $t < h_i \leq H-1$.
            Store $(s_t, a_t, s_{t+1}, q_\phi(s_{h_i}))$ into $\mathcal{R}$.
        **end for**
    **end for**
    Construct skewed replay buffer distribution $p_{\text{skewed}_{n+1}}$ using data from $\mathcal{R}$ with Equation (3.4).
    **if** total steps $< 5000$ **then**
        Fine-tune $\beta$-VAE on data uniformly sampled from $\mathcal{R}$ according to VAE Training Schedule.
    **else**
        Fine-tune $\beta$-VAE on data uniformly sampled from $\mathcal{R}$ according to VAE Training Schedule.
        Fine-tune $\beta$-VAE on data sampled from $p_{\text{skewed}_{n+1}}$ according to VAE Training Schedule.
    **end if**
**end for**

through or into a wall, then the agent will be stopped at the wall instead.

*Ant*: A MuJoCo [206] ant environment. The observation is a 3D position and velocities, orientation, joint angles, and velocity of the joint angles of the ant (8 total). The observation space is 29 dimensions. The agent controls the ant through the joints, which is 8 dimensions. The goal is a target 2D position, and the reward is the negative Euclidean distance between the achieved 2D position and target 2D position.

*Visual Pusher*: A MuJoCo environment with a 7-DoF Sawyer arm and a small puck on a table that the arm must push to a target position. The agent controls the arm by commanding $x, y$ position for the end effector (EE). The underlying state is the EE position, $e$ and puck position $p$. The evaluation metric is the distance between the goal and final puck positions. The hand goal/state space is a 10x10 $cm^2$ box and the puck goal/state space is a 30x20 $cm^2$ box. Both the hand and puck spaces are centered around the origin. The action space ranges in the interval $[-1, 1]$ in the x and y dimensions.

*Visual Door*: A MuJoCo environment with a 7-DoF Sawyer arm and a door on a table that the arm must pull open to a target angle. Control is the same as in *Visual Pusher*. The evaluation metric is the distance between the goal and final door angle, measured in radians. In this environment, we do not reset the position of the hand or door at the end of each trajectory. The state/goal space is a 5x20x15 $cm^3$ box in the $x, y, z$ dimension respectively for the arm and an angle between $[0, .83]$ radians. The action space ranges in the interval $[-1, 1]$ in the x, y and z dimensions.

*Visual Pickup*: A MuJoCo environment with the same robot as *Visual Pusher*, but now with a different object. The object is cube-shaped, but a larger intangible sphere is overlaid on top so that it is easier for the agent to see. Moreover, the robot is constrained to move in 2 dimension: it only controls the $y, z$ arm positions. The $x$ position of both the arm and the object is fixed. The evaluation metric is the distance between the goal and final object position. For the purpose of evaluation, 75% of the goals have the object in the air and 25% have the object on the ground. The state/goal space for both the object and the arm is 10cm in the $y$ dimension and 13cm in the $z$ dimension. The action space ranges in the interval $[-1, 1]$ in the $y$ and $z$ dimensions.

*Real World Visual Door*: A Rethink Sawyer Robot with a door on a table. The arm must pull the door open to a target angle. The agent controls the arm by commanding the $x, y, z$ velocity of the EE. Our controller commands actions at a rate of up to 10Hz with the scale of actions ranging up to 1cm in magnitude. The underlying state and goal is the same as in *Visual Door*. Again we do not reset the position of the hand or door at the end of each trajectory. We obtain images using a Kinect Sensor. The state/goal space for the environment is a 10x10x10 $cm^3$ box. The action space ranges in the interval $[-1, 1]$ (in cm) in the x, y and z dimensions. The door angle lies in the range $[0, 45]$ degrees.

### C.2.5 Goal-Conditioned Reinforcement Learning Minimizes $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$

Some goal-conditioned RL methods such as Warde-Farley et al. [217], Nair et al. [155] present methods for minimizing a lower bound for $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$, by approximating $\log p(\mathbf{G} \mid \mathbf{S})$ and using it as the reward. Other goal-conditioned RL methods [104, 138, 191, 3, 172, 67] are not developed with the intention of minimizing the conditional entropy $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$. Nevertheless, one can see that goal-conditioned RL generally minimizes $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$ by noting that the optimal goal-conditioned policy will deterministically reach the goal. The corresponding conditional entropy of the goal given the state, $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$, would be zero, since given the current state, there would be no uncertainty over the goal (the goal must have been the current state since the policy is optimal). So, the objective of goal-conditioned RL can be interpreted as finding a policy such that $\mathcal{H}(\mathbf{G} \mid \mathbf{S}) = 0$. Since zero is the minimum value of $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$, then goal-conditioned RL can be interpreted as minimizing $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$.

# Appendix D

# Chapter 4 Appendix

## D.1 Section 4.1 Appendix

### D.1.1 Experiment Details

In this section, we detail the experimental setups used in our results.

#### D.1.1.1 Goal State and $t$ Sampling Strategy

While $Q$-learning is valid for any value of $\mathbf{g}$ and $t$ for each transition tuple $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$, the way in which these values are sampled during training can affect learning efficiency. Some potential strategies for sampling $\mathbf{g}$ are: (1) uniformly sample future states along the actual trajectory in the buffer (i.e., for $\mathbf{s}_t$, choose $\mathbf{g} = \mathbf{s}_{t+k}$ for a random $k > 0$) as in [4]; (2) uniformly sample goal states from the replay buffer; (3) uniformly sample goals from a uniform range of valid states. We found that the first strategy performed slightly better than the others, though not by much. In our experiments, we use the first strategy. The horizon $t$ is sampled uniformly at random between 0 and the maximum horizon $t_{\max}$.

#### D.1.1.2 Model-free setup

In all our experiments, we used DDPG [138] as the base off-policy model-free RL algorithm for learning the TDMs $Q(\mathbf{s}, \mathbf{a}, \mathbf{g}, t)$. Experience replay [149] has size of 1 million transitions, and the soft target networks [138] are used with a polyak averaging coefficient of 0.999 for DDPG and TDM and 0.95 for HER and DDPG-Sparse. For HER and DDPG-Sparse, we also added a penalty on the tanh pre-activation, as in Andrychowicz et al. [4]. Learning rates of the critic and the actor are chosen from {1e-4, 1e-3} and {1e-4, 1e-3} respectively. Adam [115] is used as the base optimizer with default parameters except the learning rate. The batch size was 128. The policies and networks are parmaeterized with neural networks with ReLU hidden activation and two hidden layers of size 300 and 300. The policies have a tanh output activation, while the critic has no output activation (except for TDM, see

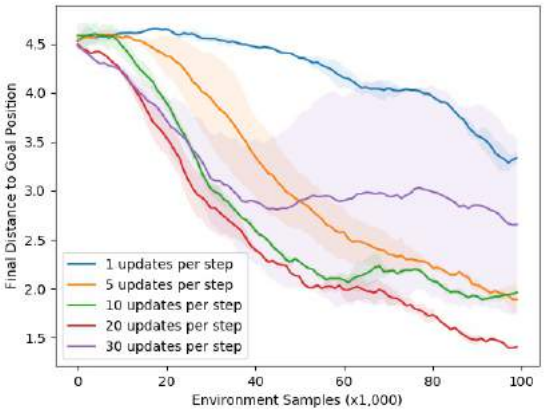



**Figure D.1:** TDMs with different number of updates per step $I$ on ant target position task. The maximum distance was set to 5 rather than 6 for this experiment, so the numbers should be lower than the ones reported in the paper.

D.1.1.5). For the TDM, the goal was concatenated to the observation. The planning horizon $t$ is also concatenated as an observation and represented as a single integer. While we tried various representations for $t$ such as one-hot encodings and binary-string encodings, we found that simply providing the integer was sufficient.

While any distance metric for the TDM reward function can be used, we chose L1 norm $-\|\mathbf{s}_{t+1} - \mathbf{g}\|_1$ to ensure that the scalar and vectorized TDMs are consistent.

#### D.1.1.3 Model-based setup

For the model-based comparison, we trained a neural network dynamics model with ReLU activation, no output activation, and two hidden units of size 300 and 300. The model was trained to predict the difference in state, rather than the full state. The dynamics model is trained to minimize the mean squared error between the predicted difference and the actual difference. After each state is observed, we sample a minibatch of size 128 from the replay buffer (size 1 million) and perform one step of gradient descent on this mean squared error loss. Twenty rollouts were performed to compute the (per-dimension) mean and standard deviation of the states, actions, and state differences. We used these statistics to normalize the states and actions before giving them to the model, and to normalize the state differences before computing the loss. For MPC, we simulated 512 random action sequences of length 15 through the learned dynamics model and chose the first action of the sequence with the highest reward.

#### D.1.1.4 Tuned Hyperparameters

For TDMs, we found the most important hyperparameters to be the reward scale, $t_{\max}$, and the number of updates per observations, $I$. As shown in Figure D.1, TDMs can greatly

benefit from larger values of $I$, though eventually there are diminishing returns and potentially negative impact, mostly likely due to over-fitting. We found that the baselines did not benefit, except for HER which did benefit from larger $I$ values. For all the model-free algorithms (DDPG, DDPG-Sparse, HER, and TDMs), we performed a grid search over the reward scale in the range $\{0.01, 1, 100, 10000\}$ and the number of updates per observations in the range $\{1, 5, 10\}$. For HER, we also tuned the weight given to the policy pre-tanh-activation $\{0, 0.01, 1\}$, which is described in [4]. For TDMs, we also tuned the best $t_{\max}$ in the range $\{15, 25, \text{Horizon} - 1\}$. For the half cheetah task, we performed extra searches over $t_{\max}$ and found $t_{\max} = 9$ to be effective.

#### D.1.1.5 TDM Network Architecture and Vector-based Supervision

For TDMs, since we know that the true $Q$-function must learn to predict (negative) distances, we incorporate this prior knowledge into the $Q$-function by parameterizing it as $Q(\mathbf{s}, \mathbf{a}, \mathbf{g}, t) = -\|f(\mathbf{s}, \mathbf{a}, \mathbf{g}, t) - \mathbf{g}\|_1$. Here, $f$ is a vector outputted by a feed-forward neural network and has the same dimension as the goal. This parameterization ensures that the $Q$-function outputs non-positive values, while encouraging the $Q$-function to learn what we call a goal-conditioned model: $f$ is encouraged to predict what state will be reached after $t$, when the policy is trying to reach goal $\mathbf{g}$ in $t$ time steps.

For the $\ell_1$ norm, the scalar supervision regresses

$$Q(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}, t) = -\sum_j |f_j(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}, t) - \mathbf{g}_j|$$

onto

$$\begin{aligned} & r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{g}) + \mathbb{1}[t = 0] + Q(\mathbf{s}_{t+1}, \mathbf{a}^*, \mathbf{g}, t-1)\mathbb{1}[t \neq 0] \\ = & -\sum_j \{|\mathbf{s}_{t+1,j} - \mathbf{s}_{g,j}|\mathbb{1}[t = 0] + |f_j(\mathbf{s}_t, \mathbf{a}^*, \mathbf{g}, t-1) - \mathbf{s}_{g,j}|\mathbb{1}[t \neq 0]\} \end{aligned}$$

where $\mathbf{a}^* = \text{argmax}_{\mathbf{a}} Q(\mathbf{s}_{t+1}, \mathbf{a}, \mathbf{g}, t-1)$. The vectorized supervision instead supervises each components of $f$, so that

$$|f_j(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}, t) - \mathbf{s}_{g,j}|$$

regresses onto

$$|\mathbf{s}_{t+1,j} - \mathbf{s}_{g,j}|\mathbb{1}[t = 0] + |f_j(\mathbf{s}, \mathbf{a}^*, \mathbf{g}, t-1) - \mathbf{s}_{g,j}|\mathbb{1}[t \neq 0]$$

for each dimension $j$ of the state.

#### D.1.1.6 Task and Reward Descriptions

Benchmark tasks are designed on MuJoCo physics simulator [206] and OpenAI Gym environments [20]. For the simulated reaching and pushing tasks, we use (4.7) and for the

other tasks we use (4.8) for policy extraction. The horizon (length of episode) for the pusher and ant tasks are 50. The reaching tasks has a horizon of 100. The half-cheetah task has a horizon of 99.

*7-DoF reacher.*: The state consists of 7 joint angles, 7 joint angular velocities, and 3 XYZ observation of the tip of the arm, making it 17 dimensional. The action controls torques for each joint, totally 7 dimensional. The reward function during optimization control and for the model-free baseline is the negative Euclidean distance between the XYZ of the tip and the target XYZ. The targets are sampled randomly from all reachable locations of the arm at the beginning of each episode. The robot model is taken from the striker and pusher environments in OpenAI Gym MuJoCo domains [20] and has the same joint limits and physical parameters.

Many tasks can be solved by expressing a desired goal state or desired goal state components. For example, the 7-Dof reacher solves the task when the end effector XYZ component of its state is equal to the goal location, $(x^*, y^*, z^*)$. One advantage of using a goal-conditioned model $f$ as in Equation (4.7) is that this desire can be accounted for directly: if we already know the desired values of some components in $\mathbf{s}_{t+T}$, then wen can simply fix those components of $\mathbf{s}_{t+T}$ and optimize over the other dimensions. For example for the 7-Dof reacher, the optimization problem in Equation (4.7) needed to choose an action becomes

$$\mathbf{a}_t = \underset{\mathbf{a}_t, \mathbf{s}_{t+T}[0:14]}{\arg\max} \; r_c(f(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+T}[0:14] || [x^*, y^*, z^*]))$$

where $||$ denotes concatenation; $\mathbf{s}_{t+T}[0:14]$ denotes that we only optimize over the first 14 dimensions (the joint angles and velocities), and we omit $\mathbf{a}_{t+T}$ since the reward is only a function of the state. Intuitively, this optimization chooses whatever goal joint angles and joint velocities make it easiest to reach $(x^*, y^*, z^*)$. It then chooses the corresponding action to get to that goal state in $T$ time steps. We implement the optimization over $\mathbf{s}[0:14]$ with stochastic optimization: sample 10,000 different vectors and choose the best value. Lastly, instead of optimizing over the actions, we use the policy trained in DDPG to choose the action, since the policy is already trained to choose an action with maximum Q-value for a given state, goal state, and planning horizon. We found this optimization scheme to be reliable, but any optimizer can be used to solve Equation (4.7),(4.6), or (4.5).

*Pusher*: The state consists of 3 joint angles, 3 joint angular velocities, the XY location of the hand, and the XY location of the puck. The action controls torques for each of the 3 joints. The reward function is the negative Euclidean distance between the puck and the puck. Once the hand is near (with 0.1) of the puck, the reward is increased by 2 minus the Euclidean distance between the puck and the goal location. This reward function encourages the arm to reach the puck. Once the arm reaches the puck, bonus reward begins to have affect, and the arm is encouraged to bring the puck to the target.

As in the 7-DoF reacher, we set components of the goal state for the optimal control formulation. Specifically, we set the goal hand position to be the puck location. To copy the two-stage reward shaping used by our baselines, the goal XY location for the puck is initially its current location until the hand reaches the puck, at which point the goal position for the

puck is the target location. There are no other state dimensions to optimize over, so the optimal control problem is trivial.

*Half-Cheetah*: The environment is the same as in [20]. The only difference is that the reward is the $\ell$-1 norm between the velocity and desired velocity $v^*$. Our optimal control formulation is again trivial since we set the goal velocity to be $v^*$. The goal velocity for rollout was sampled uniformly in the range $[-6, 6]$. We found that the resulting TDM policy tends to "jump" at the last time step, which is the type of behavior we would expect to come out of this finite-horizon formulation but not of the infinite-time horizon of standard model-free deep RL techniques.

*Ant*: The environment is the same as in [20], except that we lowered the gear ratio to 30 for all joints. We found that this prevents the ant from flipping over frequently during the initially phase of training, allowing us to run all the experiments faster. The reward is the $\ell$-1 norm between the actual and desired xy-position and xy-velocity (for the position and velocity task) of the torso center of mass. For the target-position task, the target position was any position within a 6-by-6 square. For the target-position-and-velocity task, the target position was any position within a 1-by-1 square and any velocity within a 0.05-by-0.05 velocity-box. When computing the distance for the position-and-velocity task, the velocity distance was weighted by 0.9 and the position distance was weighted by 0.1.

*Sawyer Robot*: The state and action spaces are the same as in the 7-DoF simulated robot except that we also included the measured torques as part of the state space since these can different from the applied torques. The reward function used is also the $\ell_1$ norm to the desired XYZ position.

## D.2 Section 4.2 Appendix

### D.2.1 Additional Experiments

#### D.2.1.1 Norm Ablation

We compare using the $\ell_\infty$-norm to minimize the feasibility vector with using the $\ell_1$-norm. As shown in Figure D.2, $\ell_\infty$-norm performs better, which matches the intuition it will more consistently push all terms in the feasibility vector towards zero.

#### D.2.1.2 Optimizer Ablation

We compare the performance of different optimizers on the 2D Navigation tasks. As shown in Figure D.3, CEM consistently outperforms other optimizers both in terms of the optimizer loss, and the corresponding final performance on the task.

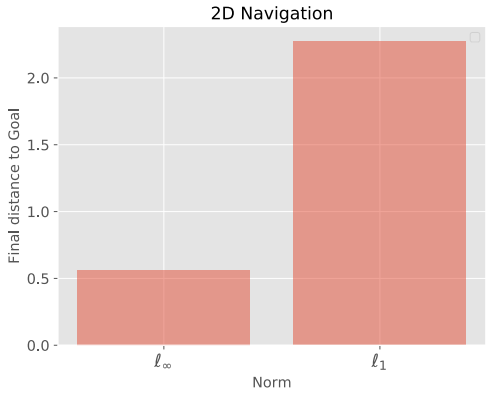



**Figure D.2:** We compare using the $\ell_\infty$-norm to the $\ell_1$-norm. We see that the $\ell_\infty$-norm outperforms the $\ell_1$-norm.

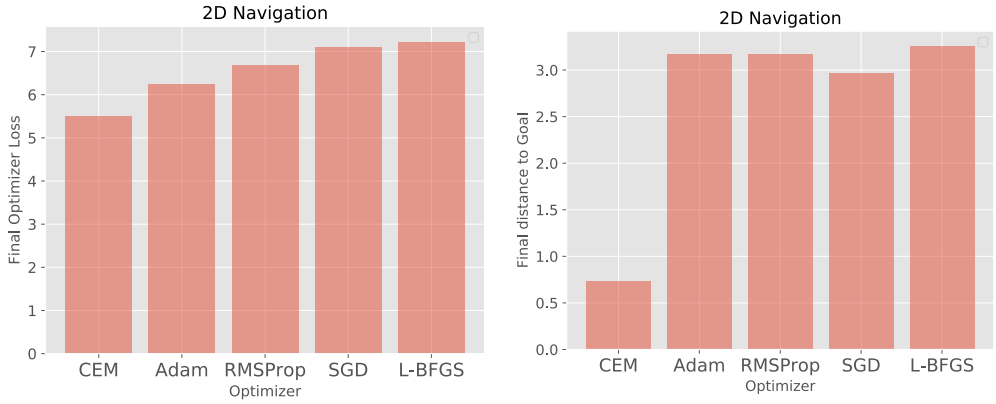



**Figure D.3:** We compare CEM to different optimizers L-BFGS, Adam, RMSProp, and gradient descent (SGD) that have had their learning rates tuned. (Left) The optimizer loss, where CEM outperforms the other methods. (Right) The performance of the policy after using the plan chosen by each optimizer. We see that the lower optimizer loss of CEM corresponds to a better performance.

### D.2.1.3 Likelihood Penalty Ablation

We examine the effect of the additional log-likelihood term (under the VAE prior) in Equation (4.10). In particular, we vary the weighting hyperparameter $\lambda$ for the 2D Navigation and Push and Reach environments. For each environment, we note the final performance of the RL algorithm, in addition to the log-likelihood values and V values that compose (4.10). See Figure D.4 for detailed results. We see that there is a trade-off between achieving a high likelihood under the prior and high V values. As we increase the weighting term $\lambda$ the likelihood values increase while the V values decrease. There is an optimal threshold at which RL performance is maximized. For 2D Navigation, we note this value to be $\lambda = 0.01$ and for Push and Reach any range of values between 0.0001 and 0.01. For Ant Navigation, we independently verified an optimal choice of $\lambda = 0.1$.

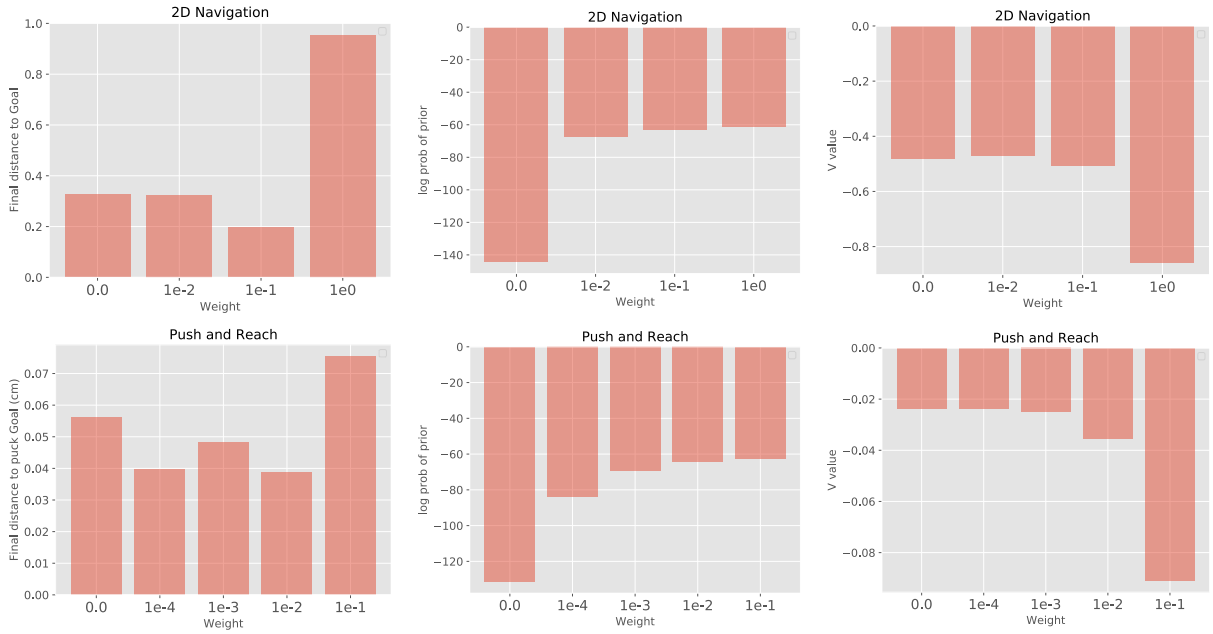



**Figure D.4:** Examining the effect of the weight $\lambda$ in Equation 4.10. We note the final RL performance (left), log-likelihood under the VAE prior (middle), and V values (right). As we increase $\lambda$, the log-likelihood values increase while the V values decrease. For 2D navigation (top), we note the optimal value to be $\lambda = 0.01$ and for Push and Reach (bottom) any range of values between 0.0001 and 0.01.

## D.2.2 Environment Details

### D.2.2.1 2D Navigation

The agent must learn to navigate around a square room with a U-shaped wall in the center. See Figure 4.6 for a visualization of the environment. The dimensions of the space are $8 \times 8$ units, the walls are 1 unit thick, and the agent is a circle with radius 0.5 units. The observation is a $48 \times 48$ RGB image and the agent specifies a 2D velocity as the action. At each timestep, the agent can attempt to move up to 0.15 units in either dimension. The distance for subsubsection 4.1.2.1 is the distance between the current 2D position and the target position. We note that a greedy policy can easily lower the final distance by moving directly towards the goal. To measure whether or not the final policy performs more non-greedy behavior, we define success as whether or not the policy ends below the horizontal wall and within a diameter of the intended goal. Complete results are provided in Figure D.5. Plots are averaged across 5 seeds, with the exception of PETS, which uses 3 seeds due to computational constraints. For image based baselines (all except PETS), we first train VAEs and select the top 5 seeds based on VAE loss. We proceed to training our RL algorithm with one seed per selected VAE. Note that for the ablation study in Figure 4.9, we select the top VAE seed based on VAE loss, and train our RL algorithm with 5 seeds.

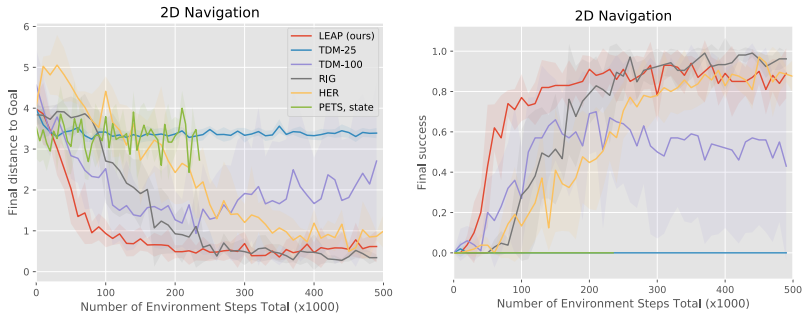



**Figure D.5:** Complete 2D Navigation Results

#### D.2.2.2 Push and Reach

This task is based on the environment released by Nair et al. [155]. An additional invisible wall around the goal space of the puck has been added to prevent the puck from moving to unreachable hand locations. In contrast to prior work evaluated on goal-conditioned pushing tasks [3, 171, 35], this task is solved using images as the observations and cannot be solved with a simple, unidirectional pushing behavior [155, 173]. Specifically, the observation is an $84 \times 84$ RGB image showing a top-down view of the scene. The robot is operated via 2D position control, where each action is limited to moving the robot end effector 2 cm in either dimension. The distance for Section 4.1.2.1 is the Euclidean distance between (1) the goal and (2) the XY-position of the puck concatenated with the XY-position of the hand. We modify the task so as to require the agent to perform temporally extended planning. First, we increase the workspace of the environment to 40 cm $\times$ 20 cm. Second, we evaluate the final policy on 5 hard scenarios which require temporally extended behavior: rather than simply executing a simple, unidirectional pushing behavior, the robot must reach across the table to a corner where the puck is located, move its arm around the puck, and then pull the puck to a different corner of the table, as shown in Figure 4.6. A trajectory is successful if the final puck position is within 6 cm of the target position. For context, the puck has a radius of 4 cm. Complete results are provided in Figure D.6. Plots are averaged across 8 seeds, with the exception of PETS, which uses 5 seeds due to computational constraints. For image based baselines (all except PETS), we first train VAEs and select the top 8 seeds based on VAE loss. We proceed to training our RL algorithm with one seed per selected VAE.

#### D.2.2.3 Ant Navigation

The ant must learn to navigate around a narrow rectangular room with a long wall in the center. See Figure 4.8 for a visualization of the environment. The dimensions of the space are $7.5 \times 18$ units, the wall is 1.5 units thick, and the ant has a radius of roughly 0.75 units. The state includes the position, orientation (in Euler angles rather than quaternions), joint angles, and velocities of the aforementioned components. The gear ratio for the ant is reduced to 10 units, to prevent the ant from flipping over. The distance for subsubsection 4.1.2.1 is the distance between the current 2D position and the target position, in addition to the

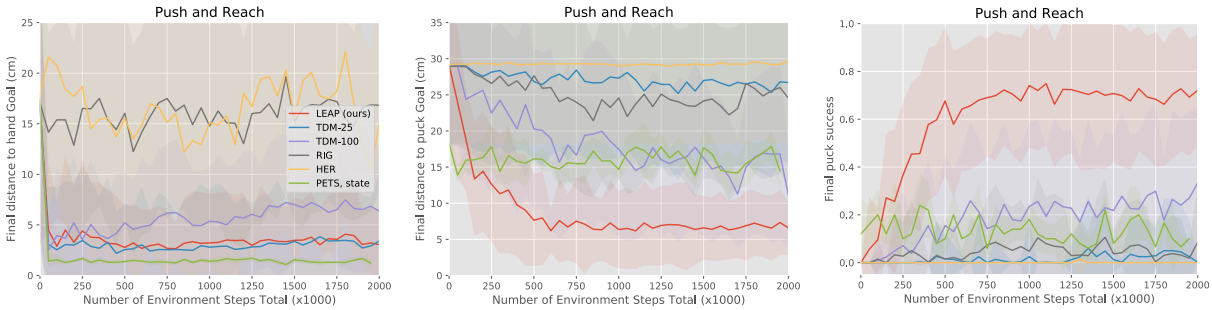



**Figure D.6:** Complete Push and Reach Results

differences in orientation of the ant with respect to the target orientation. We define success as whether or not the ant is within 1.5 units of the goal position. Complete results are provided in Figure D.7. Plots are averaged across 15 seeds, with the exception of HIRO, which uses 5 seeds due to computational constraints. For LEAP, we first train VAEs and select the top 5 seeds based on VAE loss. We proceed to training our RL algorithm with three seed per selected VAE. Unlike the image-based experiments, the VAE is not used for training the RL algorithm. It is only used during test time for planning subgoals. The VAE is trained on a dataset in which the ant is in various valid positions of the maze, with a fixed orientation and fixed joint angles.

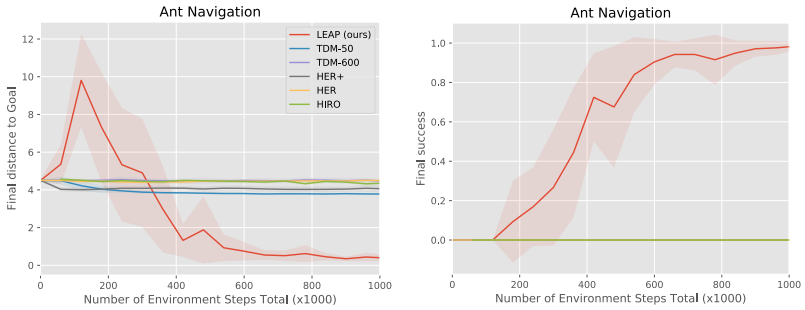



**Figure D.7:** Complete Ant Navigation Results

## D.2.3 Implementation Details

This section contains descriptions and hyperparameters of the experiment implementations.

### D.2.3.1 Goal-conditioned reinforcement learning

Both the $Q$ network and policy concatenate all inputs and pass them through a feed-forward network. For RIG, the Q network outputs a scalar corresponding to the infinite discounted sum of rewards. For TDMs, the Q network outputs a vector corresponding to the negative distance between the final state and goal along each of the state dimensions. We train our

| Hyper-parameter | Value |
|---|---|
| Q network hidden sizes | $400, 300$ |
| Policy network hidden sizes | $400, 300$ |
| Q network and policy activation | ReLU |
| Q network output activation | None |
| Policy network output activation | tanh |
| Exploration noise | $\epsilon$-greedy, $\epsilon = .1$ (2D Navigation)<br>OU-process $\theta = .3$, $\sigma = .3$ (Push & Reach and Ant) |
| # training batches per time step | 1 |
| Batch size | 128 (2D Navigation)<br>2048 (Push and Reach and Ant) |
| Optimizer | Adam |
| Learning rate (all networks) | 0.001 |
| Target update rate $\tau$ | 0.005 |
| Replay buffer size | 1000000 |

**Table D.1:** TD3 [71] hyperparameters.

networks using the twin delayed deep deterministic policy gradient algorithm [71] (TD3). Hyperparameter details are provided in Table D.1. When sampling minibatches from the replay buffer, we sample transitions, goals, and times (for TDMs only). For TDM, RIG, and HER+, we relabel the goals in our minibatches in the following manner:

- 20%: original goals from collected trajectories
- 40%: randomly sampled states from the replay buffer
- 40%: future states along the same collected trajectory, as dictated by hindsight experience replay [3] (HER).

We note that in the Ant Navigation task, we split sampling from the replay buffer to 20% from the replay buffer and 20% oracle goals from the environment.

For HER, we relabel the goals in our minibatches in the following manner:

- 20%: original goals from collected trajectories
- 80%: future states along the same collected trajectory

#### D.2.3.2 Latent space optimization

In this subsection, we describe how we use the cross entropy method (CEM) [42] to optimize (4.10). Given an optimization problem over $K$ subgoals, with each subgoal represented as an $r$-dimensional latent vector, the CEM optimizer is initialized with a standard multivariate Gaussian distribution $\mathcal{N}(0_{rK}, I_{rK})$, where $0_{rK}$ is a $rK$-dimensional vector of zeros, and $I_{rK}$ is the $rK \times rK$ identity matrix. We sample different subgoal sequences from our distribution

and evaluate the value of each sample using Equation 4.10. We then fit a diagonal multivariate Gaussian distribution to the top 5% of samples. We repeat this process for 15 iterations, and at each iteration we sample 1000 subgoal sequences from the fitted Gaussian. For the Ant Navigation task which involves optimizing over significantly higher number of subgoals, we sample 10000 subgoal sequences and run for 50 iterations instead. In addition, we found it beneficial to filter the top 25% of samples for the first half of iterations, and then filter the top 1% in the latter half. For the weight on the log-likelihood of the latents, we use $\lambda = 0.1$ for 2D Navigation and Ant Navigation tasks, and $\lambda = 0.001$ for Push and Reach.

#### D.2.3.3 Variational auto-encoder

We use separate VAE architectures for 2D Navigation ($48 \times 48$ image) and Push and Reach ($84 \times 84$ image). For 2D Navigation, encoder kernel sizes of $[5, 3, 3]$, encoder strides of $[3, 2, 2]$, $[16, 32, 64]$ encoder channels, decoder kernel sizes of $[3, 3, 6]$, decoder strides of $[2, 2, 3]$, and $[64, 32, 16]$ decoder channels are used. For Push and Reach, we use encoder kernel sizes of $[5, 5, 5]$, encoder strides of $[3, 3, 3]$, $[16, 16, 32]$ encoder channels, decoder kernel sizes of $[5, 6, 6]$, decoder strides of $[3, 3, 3]$, and $[32, 32, 16]$ decoder channels. Both architectures have a representation size of 16 and ReLU activation. We trained the 2D Navigation VAEs with binary cross-entropy loss, and the Push and Reach VAEs with mean squared error loss.

For Ant Navigation, our VAE is a generative model for the full state of the ant, rather than images. Our encoder and decoder are multilayer perceptrons with hidden sizes of $[64, 128, 64]$ and ReLU activation. We used a representation size of 8, and trained the VAE with mean squared error loss.

# Appendix E

# Chapter 5 Appendix

## E.1 Section 5.1 Appendix

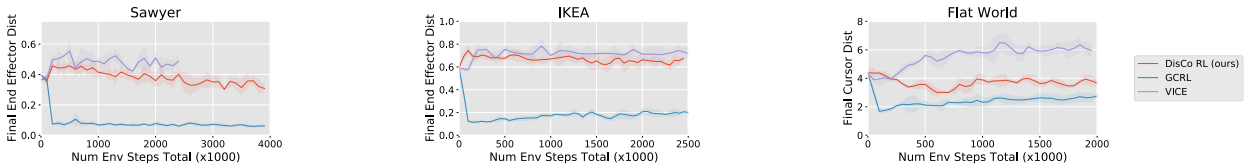
### E.1.1 Additional Results

The objective of the tasks in Section 5.1.4 was to move objects to certain positions, and we found that goal-conditioned reinforcement learning (GCRL) performed poorly. To better understand the behavior of the GCRL policies, we plotted the distance between the final end effector position reached and the one specified by the final goal state $\mathbf{s}_g$. In Figure E.1, we see that the GCRL policies quickly learned to minimize the final distance to the end effector position in the goal state $\mathbf{s}_g$. We see that the GCRL policies optimized only the dimensions of the goal state that were easiest to maximize, rather than the important dimensions. VICE had unsatisfactory performance even on the end effector position, and we hypothesize this is due to the fact that VICE needs a substantially higher number of example states.

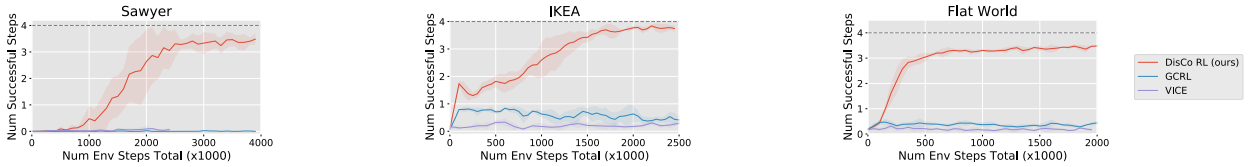
Videos of the final policies for our method and baselines are available on the paper website: `https://sites.google.com/view/disco-rl`

### E.1.2 Environments

**Sawyer** This environment is based in the PyBullet [39] physics simulator. It consists of a Sawyer robot mounted next to a table, on top of which there is a tray and four blocks. The robot must learn to manipulate the blocks via its gripper. The robot is controlled via position control, and it is restricted to move in a 2D plane. Specifically, the robot arm can move in the YZ coordinate plane and the gripper can open along the X axis, where the X, Y, Z axes move along the front-back, left-right, and up-down directions of table, respectively. We also constrain the objects to move along the YZ coordinate plane. The agent has access to state information, comprising of the position of the end effector and gripper state, as well as the positions of the objects and tray.




**(a)** final end-effector distance




**(b)** success rate

**Figure E.1:** (a) The distance between the final end-effector position and the position specified in the goal state $\mathbf{s}_g$. (b) The success rate of the methods on the same task (with results copied from Figure 5.5 for convenience). We see that goal-conditioned RL focuses primarily on moving the end effector to the correct position, while DisCo RL ignores this task-irrelevant dimension and successfully completes the task.

**Visual Sawyer** We use the same environment as described above for the vision-based tasks but replace the state with 48x48 RGB images. We pretrain a VAE on 5120 randomly generated images and obtain the goal distribution using 30 example images. We visualize some example images in Figure E.2.

The encoder consist of a 3 convolutions with the following parameters

1. channels: 64, 128, 128
2. kernel size: 4, 4, 3
3. stride: 2, 2, 2
4. padding: 1, 1, 1

followed by 3 residual layers each containing two convolutions with the following parameters

1. channels: 64, 64
2. kernel size: 3, 1
3. stride: 1, 1
4. padding: 1, 1

and two linear layers that projects the convolution output into the mean and log-standard deviation of a Gaussian distribution in a latent dimension with dimension 64.

The decoder begins with a linear layer with the transposed shape as the final encoder linear layer, followed by a reshaping into a latent image the same shape as the final encoder

convolution output shape. This latent image is put through a convolution (128 channels, kernel size 3, stride 1, padding 1), and then an equivalent residual stack as the encoder, and two transposed convolutions with parameters

1. channels: 64, 64
2. kernel size: 4, 4
3. stride: 2, 1
4. padding: 2, 1

which outputs the mean of a Gaussian distribution with a fixed unit variance. We trained this VAE with Adam with a learn rate of $10^{-3}$ and default PyTorch [165] parameters ($\beta_1 = 0.9$ and $\beta_2 = 0.999$) for 100 epochs and annealed the loss on the KL term from 0 to 1 for the first 20 epochs.

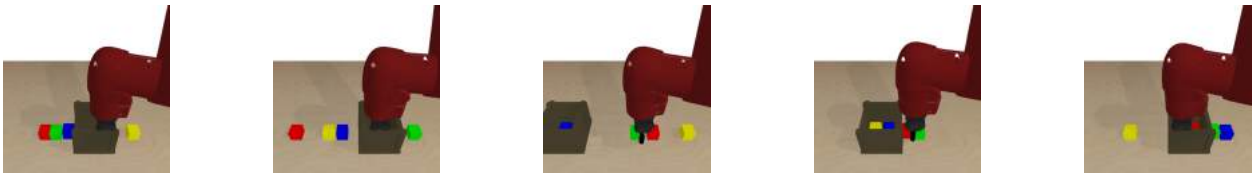


**Figure E.2:** Example images with the hand in a fixed position used to obtain a goal distribution.

**IKEA** We adapted this environment from the suite of furniture assembly environments developed by Lee et al. [131]. In our environment, the agent must learn to attach a set of 3 shelves to a pole. It can do so by controlling two end effectors: one end effector that can move the pole, and another end effector that can move the shelves. The former end effector is always attached to the pole, while the latter end effector can selectively attach and detach itself from the shelves. Both end effectors can move via 3D position control, in a $1 \times 1 \times 1$ area for a maximum of 0.05 units (in each direction) per timestep. The end effector interfacing with the shelves can hold onto a shelf by applying a grasp action when it is within the bounding box region of a shelf. Each shelf has a connection point at which point it will attach to the pole, and the pole has a receiving connection point as well. When these two connection points are within 0.2 units away from one another, the shelf automatically attaches to the pole and becomes welded. The objects have 3 degrees of freedom via translations in 3D space. The objects are not allowed to collide with one another – if an action causes them to collide, that action is ignored by the environment and the next state is the same as the current state. As an exception, when an end effector is not grabbing an object, it is allowed to move through objects. The agent has access to state information, comprising the 3D position of the end effectors, shelves, and the pole, and indicator information for whether each end effector is grasping an object.

**Flat World** This two-dimensional environment consists of a policy and 4 objects, each of which are defined by their XY-coordinate. The policy and objects are in an enclosed $8 \times 8$ unit space. The policy's action space is three dimensional: two correspond to relative change in position, for a maximum of 1 unit in each dimension per timestep, and one corresponds to a grab action. The grab action takes on a value between $-1$ and 1. If this grab action is positive, then the policy picks up the closest object that is within 1 unit of it, or none if there are no such objects. If this grab action is non-positive, then the policy drops any object that it was holding. While an object is grabbed, the object moves rigidly with the policy. The policy can only grab one object at a time, with ties broken by a predetermined, fixed order. The agent has access to state information, comprising the 2D position of the policy and the 4 objects.

### E.1.3 Experimental Details

**Distribution inference** These experiments were evaluated on the Sawyer environment. In this setting, the agent needs to pick up one of the blocks (specifically, the red block) and place it into the tray. The initial position of the tray and objects vary in each episode. The objective is only to minimize the relative distance between the red block and the tray, and it is important for the agent to ignore the absolute position of the other blocks and the tray. The robot can attempt to slide the tray to a specific goal location, but the tray is heavy and moves very slowly. If it successfully moves the tray to another location, it will not have enough time in the episode to move the red block.

We generated $K = 30$ examples of successful goal states, in which the red block is always inside the tray, and the tray, other objects, hand, and gripper are in random locations and configurations. We provided this set of example goal states as input to the competing baselines. Each baseline used the example states in the following manner:

- DisCo RL: infer a goal distribution from the example states
- VICE: a classifier is trained to predict whether a state is optimal, with the example states as the positive examples for the classifier

For evaluation, a trajectory is successful if the red block is placed in the tray. We plot this success metric over time in Figure 5.4.

**Multi-task performance** For the multi-task evaluations, we performed experiments in all three of our environments. For each environment, we split the task into several subtasks, as described below:

- Sawyer: move the blocks to their goal locations. Each subtask represents moving one block at a time to its goal location.
- IKEA: move the pole and the shelves to their goal locations. Each subtask represents moving the pole and one of the shelves to their goal locations.

- Flat World: move the objects to their goal locations. Each subtask represents moving one object at a time to its goal location.

We generated an example dataset of successful states for each subtask. For each example state for a particular subtask, we also provided a state representing the final configuration for the entire task (after all subtasks are solved). See **??** E.1.4.2 for additional details regarding the example sets. The evaluation metrics for each environment are as follows:

- Sawyer: the number of objects that are within 0.10 units of their respective final goal locations
- IKEA: the number of shelves that are connected to the pole, in addition to an indicator for whether the pole is within 0.10 units of its final goal location
- Flat World: the number of objects that are within 1 unit of their respective final goal locations

For evaluation, we provide the GCRL baseline oracle goals. In this setting, the provided goal is identical to the initial state at the beginning of the episode, except for the position of the red block, which we set to be inside the tray. For example, if the state is given by

$$\mathbf{s}_0 = \left[x_0^{\text{EE}}, y_0^{\text{EE}}, x_0^{\text{red-block}}, y_0^{\text{red-block}}, x_0^{\text{blue-block}}, y_0^{\text{blue-block}}, \ldots\right],$$

and we want the red block to move to a position $(x^*, y^*)$, we set the goal to

$$\mathbf{s}_g = \left[x_0^{\text{EE}}, y_0^{\text{EE}}, x^*, y^*, x_0^{\text{blue-block}}, y_0^{\text{blue-block}}, \ldots\right].$$

In theory, this oracle goal encourages the robot to focus on moving the red block to its goal rather than moving the other state components. The GCRL baseline only uses this oracle during evaluation.

Details regarding the distribution of final goal states used for exploration rollouts and relabeling during training, are provided in Table E.1. For evaluation, we used $H = 100$ for the IKEA and Flat World environments, and $H = 400$ for the Sawyer environment.

| **Goal Use Case** | **Sawyer** | **IKEA** | **Flat World** |
|---|---|---|---|
| Exploration | 50%: objects on ground, 50%: objects in tray | Shelves assembled, pole in random position | Objects in random positions |
| Training | Same as above | Objects in random positions | Same as above |

**Table E.1:** Environment specific final goal distributions.

| Hyper-parameter | Sawyer | IKEA | Flat World |
|---|---|---|---|
| Number of examples (per subtask) $K$ | 30 | 50 | 30 |
| Std. dev. of Gaussian noise added to example set data | 0.01 | 0.1 | 0.01 |

**Table E.2:** Environment specific hyper-parameters.

| Hyper-parameter | Value |
|---|---|
| horizon $H$ (for training) | 100 |
| batch size | 2048 |
| discount factor | 0.99 |
| $Q$-function and policy hidden sizes | $[400, 300]$ |
| $Q$-function and policy hidden activations | ReLU |
| replay buffer size | 1 million |
| hindsight relabeling probability | 80% |
| target network update speed $\tau$ | 0.001 |
| number of training updates per episode $N_{\text{updates per episode}}$ | 100 |
| number of training batches per environment step | 1 |

**Table E.3:** General hyper-parameters used for all experiments.

**Off-policy distributions** These experiments are conducted on the Flat World domain and extend the multi-task experiments described in the previous section. We created a total of 10 total subtasks, 4 of which require moving a single object to its desired goal location, and 6 of which require moving a pair of objects to their desired goal locations. We provided example sets of size $K = 30$ for each of these 10 subtasks. For the on-policy variant, we explore and relabel with the goal distributions inferred for all 10 subtasks. For the off-policy variant, we only explore with the first 4 goal distributions and train with all 10 goal distributions. For evaluation, we measured the performance of the algorithm for one of the pairwise subtasks. Our evaluation metric measures how many objects (out of the specific pair) the agent was able to successfully move within 1 unit of their respective final goal locations.

## E.1.4 Implementation Details

### E.1.4.1 General Training Algorithm and Hyperparameters

In our experiments, we use soft actor-critic as our RL algorithm [83]. For specific details on the hyperparameters that we used, see Table E.3.

### E.1.4.2 DisCo RL

**Covariance Smoothing** We apply pre-preprocessing and post-processing steps to obtain the distribution parameters used in RL. In the pre-processing phase, we add i.i.d. Gaussian noise to the dataset of examples. The amount of noise that we apply varies by environment

– see Table E.2 for specific details. After inferring the raw parameters of the Gaussian distribution $\mu$ and $\Sigma$, we apply post-processing steps to the covariance matrix. We begin by inverting the covariance matrix $\Sigma$. For numerical stability, we ensure that the condition number of $\Sigma$ does not exceed 100 by adding a scaled version of the identity matrix to $\Sigma$. After obtaining $\Sigma^{-1}$, we normalize its components such that the largest absolute value entry of the matrix is 1. Finally, we apply a regularization operation that thresholds all values of the matrix whose absolute value is below 0.25 to 0. We found this regularization operation to be helpful when the number of examples provided is low, to prevent the Gaussian model from inferring spurious dependencies in the data. We used the resulting $\mu$ and $\Sigma^{-1}$ for computing the reward.

**Conditional distribution details** To obtain the conditional distribution used for relabeling and multi stage planning, we assume that data is given in the form of pairs of states $\{(\mathbf{s}^{(k)}, \mathbf{s}_f^{(k)})\}_{k=1}^K$, in which $\mathbf{s}^{(k)}$ correspond to a state where a sub-task is accomplished when trying to reach the final state $\mathbf{s}_f$. We fit a joint Gaussian distribution of the form

$$p_{\mathbf{s},\mathbf{s}_f} = \mathcal{N}\left(\begin{bmatrix}\mu_{\mathbf{s}} \\ \mu_{\mathbf{s}_f}\end{bmatrix}, \begin{bmatrix}\Sigma_{\mathbf{ss}} & \Sigma_{\mathbf{s}\mathbf{s}_f} \\ \Sigma_{\mathbf{s}_f\mathbf{s}} & \Sigma_{\mathbf{s}_f\mathbf{s}_f}\end{bmatrix}\right), \mu_{(\cdot)} \in \mathbb{R}^{\dim(\mathcal{S})}, \Sigma_{(\cdot)} \in \mathbb{R}^{\dim(\mathcal{S})\times\dim(\mathcal{S})}$$

to these pairs of states using maximum likelihood estimation. Since the joint distribution $p_{\mathbf{s},\mathbf{s}_f}$ is Gaussian, the conditional distribution $p_{\mathbf{s}|\mathbf{s}_f}$ is also Gaussian with parameters $(\mu, \Sigma) = h(\mathbf{s}_f)$, where $h$ is the standard conditional Gaussian formula:

$$h(\mathbf{s}_f) = \left(\underbrace{\mu_{\mathbf{s}} + \Sigma_{\mathbf{s}\mathbf{s}_f}\Sigma_{\mathbf{s}_f\mathbf{s}_f}^{-1}(\mathbf{s}_f - \mu_{\mathbf{s}_f})}_{\mu}, \underbrace{\Sigma_{\mathbf{ss}} - \Sigma_{\mathbf{s}\mathbf{s}_f}\Sigma_{\mathbf{s}_f\mathbf{s}_f}^{-1}\Sigma_{\mathbf{s}_f\mathbf{s}}}_{\Sigma}\right). \tag{E.1}$$

In summary, given a final desired state $\mathbf{s}_f$, we generate a distribution by computing $\omega = h(\mathbf{s}_f)$ according to Equation (E.1). This conditional distribution also provides a simple way to relabel goal distributions given a reached state $\mathbf{s}_r$: we relabel the goal distribution by using the parameters $\omega' = h(\mathbf{s}_r)$.

**Multi-task exploration scheme** For training, in 50% of exploration rollouts we randomly selected a single subtask for the entire rollout, and in the other 50% of exploration rollouts we sequentially switched the subtask throughout the rollout, evenly allocating time to each subtask. For switching the subtask, we simply switched the parameters of the subtask $\mu$ and $\Sigma$. We randomized the order of the subtasks for sequential rollouts.

**Relabeling** We relabel the parameters of the goal distribution $(\mu, \Sigma)$, and the relabeling strategy we use depends on whether we use conditional goal distributions. For non-conditional distributions, we relabel according to the following strategy:

- 40%: relabel $\mu$ to a future state along the same collected trajectory

For conditional distributions, we relabel according to the following strategy:

- 40%: randomly sample $\mathbf{s}_f$ from the environment
- 40%: relabel $\mathbf{s}_f$ to a future state along the same collected trajectory

For our multi-task experiments, whenever we perform relabeling, we also relabel $\Sigma$. Specifically, we first randomly sample a task from the set of tasks that we have inferred, and relabel $\Sigma$ to the covariance matrix for that task.

#### E.1.4.3 HER

Our implementation of goal-conditioned RL follows from hindsight experience replay (HER) [3]. Crucially, we perform off-policy RL, in addition to using the relabeling strategies inspired by HER. When provided a batch of data to train on, we relabel the goals according to the following strategy:

- 40%: randomly sampled goals from the environment, or the example sets
- 40%: future states along the same collected trajectory, as dictated by HER

Unlike HER, which used sparse rewards, we use the Euclidean distance as the basis for our reward function:

$$r(\mathbf{s}, \mathbf{s}_g) = -\left\|\mathbf{s} - \mathbf{s}_g\right\| \tag{E.2}$$

To avoid manual engineering, the space of goals is the same as the space of states. I.e., the dimension of the goal is the same as that of the state, and the corresponding entries in $\mathbf{s}$ and $\mathbf{s}_g$ correspond to the same semantic state features.

#### E.1.4.4 VICE

Variational inverse control with events (VICE) is described in Fu et al. [70]. VICE proposes an inverse reinforcement learning method that extends adversarial inverse reinforcement learning (AIRL) Fu et al. [69]. Like AIRL, VICE learns a density $p_\theta(s, a)$ using a classification problem. However, unlike the usual IRL setting, VICE assumes access to an example set that specifies the task – the same assumption as DisCo RL.

VICE alternates between two phases: updating the reward and running RL. To learn a reward function, VICE solves a classification problem, considering the initial example set as positives and samples from the replay buffer as negatives. The discriminator is:

$$D_\theta(\mathbf{s}, \mathbf{a}) = \frac{p_\theta(\mathbf{s}, \mathbf{a})}{p_\theta(\mathbf{s}, \mathbf{a}) + \pi(\mathbf{a} \mid \mathbf{s})}. \tag{E.3}$$

At optimality, the reward recovered $p_\theta(\mathbf{s}, \mathbf{a}) \propto \pi^*(\mathbf{a} \mid \mathbf{s}) = \exp(A(\mathbf{s}, \mathbf{a}))$, the advantage of the optimal policy [69]. In practice the reward is represented as $p_\theta(\mathbf{s})$, ignoring the dependence

on actions. However, actions are still needed to compute the discriminator logits; we follow the method specified in VICE-RAQ [196] to sample actions from $\pi(\mathbf{a} \mid \mathbf{s})$ for all states. We also use mixup [228] as described in VICE-RAQ. Mixup significantly reduces overfitting and allows VICE to successfully learn a neural net classifier even with so few (30 to 50) positive examples. In the RL phase, VICE runs reinforcement learning with $\log p_\theta(\mathbf{s}, \mathbf{a})$ as the reward function, actively collecting more samples to use as negatives.

We re-implemented VICE as above and confirmed that it successfully learns policies to reach a single state, specified by examples. However, our results demonstrate that VICE struggles to reach the example sets we use in this work. This issue is exacerbated when the problem is multi-task instead of single-task and the state includes a goal, as in goal-conditioned learning.

In multi-stage tasks, we train a single DisCo RL policy shared among the stages. For VICE, sharing data among different tasks would not respect the adversarial optimization performed by the method. Instead, we train separate policies for each stage without sharing data between policies. Thus, for a task with $N$ stages, VICE is generously allowed to experience $N\times$ the data, as each policy collects its own experience).

# E.2 Section 5.2 Appendix

## E.2.1 Additional Experimental Results

**Exploration and offline dataset visualization** In Figure E.3, we visualize the post-adaption trajectories generated when conditioning the encoder the online exploration trajectories $\mathbf{h}_{\text{online}}$ and the offline trajectories $\mathbf{h}_{\text{offline}}$, similar to Figure 5.11, and also visualize the online and offline trajectories themselves. We see that $\mathbf{h}_{\text{online}}$ and $\mathbf{h}_{\text{offline}}$ are very different, but the self-supervised phase mitigates the negative impact that this distribution shift has on offline meta RL.

## E.2.2 Experimental Details

### E.2.2.1 Environment Details

In this section, we describe the state and action space of each environment. We also describe how reward functions were generated and how the offline data was generated.

**Ant Direction** The Ant Direction task consists of controlling a quadruped "ant" robot that can move in a plane. Following prior work [176, 50], the reward function is the dot product between the agent's velocity and a direction uniformly sampled from the unit circle. The state space is $\mathbb{R}^{20}$, comprising the orientation of the ant (in quaternion) as well as the angle and angular velocity of all 8 joints. The action space is $[-1, 1]^8$, with each dimension corresponding to the torque applied to a respective joint.

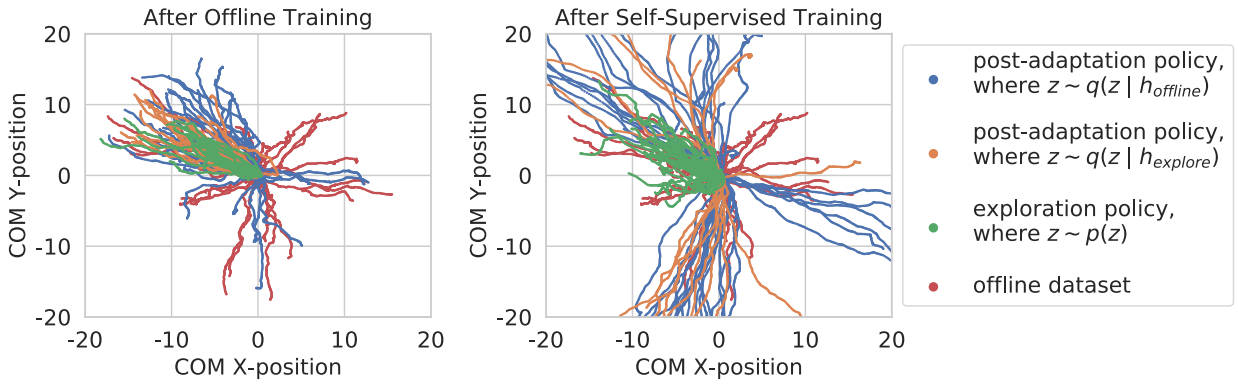



**Figure E.3:** We duplicate Figure 5.11 but include the exploration trajectories (green) and example trajectories from the offline dataset (red). We see that the exploration policy both before and after self-supervised training primarily moves up and to the left, whereas the offline data moves in all direction. Before the self-supervised phase, we see that conditioning the encoder on online data (orange) rather than offline data (blue) results in very different policies, with the online data resulting in the post-adaptation policy only moving up and to the left. However, the self-supervised phase of SMAC mitigates the impact of this distribution shift and results in qualitatively similar post-adaptation trajectories, despite the large difference between the exploration trajectories and offline dataset trajectories.

The offline data is collected by running PEARL [176] on this meta RL task with 100 pre-sampled [1] target velocities. We terminate PEARL after 100 iterations, with each iteration containing at least 1000 new transitions. In PEARL, there are two replay buffers saved for each task, one for sampling data for training the encoder and another for training the policy and $Q$-function. We will call the former replay buffer the encoder replay buffer and the latter the RL replay buffer. The encoder replay buffer contains data generated by only the exploration policy, in which $\mathbf{z} \sim p(\mathbf{z})$. The RL replay buffer contains all data generated, including both exploration and post-adaptation, in which $\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{h})$. To make the offline dataset, we load the last 1200 samples of the RL replay buffer and the last 400 transitions from the encoder replay buffer into corresponding RL and encoder replay buffers for SMAC. In the initial submission, we mistakenly stated that we used 1200 samples when in fact we used 1600 samples for each task. During the self-supervised phase, we add all new data to both replay buffers.

**Cheetah Velocity** The Cheetah Velocity task consists of controlling a two-legged "half cheetah" that can move forwards or backwards along the x-axis. Following prior work [176, 50], the reward function is the absolute difference product between the agent's x-velocity and a velocity uniformly sampled from $[0, 3]$. The state space is $\mathbb{R}^{20}$, comprising the $z$-position; the cheetah's x- and z- velocity; the angle and angular velocity of each joint and the half-cheetah's

[1] To mitigate variance coming from this sampling procedure, we use the same sampled target velocities across all experiments and comparisons. We similarly use a pre-sampled set of tasks for the other environments.

y-angle; and the XYZ position of the center of mass. The action space is $[-1, 1]^6$, with each dimension corresponding to the torque applied to a respective joint.

The offline data is collected in the same way as in the Ant Direction task, using a run from PEARL with 100 pre-sampled target velocities. For the offline dataset, we use the first 1200 samples from the RL replay buffer and last 400 samples from the encoder replay buffer after 50 PEARL iterations, with each iteration containing at least 1000 new transitions. For only this environment, we found that it was beneficial to freeze the encoder buffer during the self-supervised phase.

**Sawyer Manipulation** The state space, action space, and reward is described in Section 5.2.5. Tasks are generated by sampling the initial configuration, and then the desired behavior. There are five objects: a drawer opened by handle, a drawer opened by button, a button, a tray, and a graspable object. If an object is not present, it takes on position 0 in the corresponding element of the state space. First, the presence or absence of each of the five is randomized. Next, the position of the drawers (from 2 sides), initial position of the tray (from 4 positions), and the object (from 4 positions) is randomized. Finally, the desired behavior is randomly chosen from the following list, but only including the ones that are possible in the scene: "move hand", "move object", opening or closing each of the 2 drawers (4 tasks in total), "press button", and "move object to tray". The offline data is collected using a scripted controller that does not know the desired behavior and randomly performs potential tasks in the scene, choosing another task if it finishes one task before the trajectory ends. This data is loaded into a single replay buffer used for both the encoder and RL.

#### E.2.2.2 Hyperparameters

We list the hyperparameters for training the policy, encoder, decoder, and Q-network in Table E.4. If hyperparameters were different across environments, they are listed in Table E.5. For pretraining, we use the same hyperparameters and train for 50000 gradient steps. Below, we give details on non-standard hyperparameters and architectures.

**Batch sizes.** The RL batch size is the batch size per task when sampling $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ tuples to update the policy and Q-network. The encoder batch size is the size of the history $\mathbf{h}$ per task used to conditioned the encoder $q_\phi(\mathbf{z} \mid \mathbf{h})$. The meta batch size is how many tasks batches were sampled and concatenated for both the RL and encoder batches. In other words, for each gradient update, the policy and Q-network observe (RL batch size) $\times$ (meta batch size) transitions and the encoder observes (RL batch size) $\times$ (encoder batch size) transitions.

**Encoder architecture.** The encoder uses the same architecture as in Rakelly et al. [176]. The posterior is given as the product of independent factors

$$q_\phi(\mathbf{z} \mid \mathbf{h}) \propto \prod_{\mathbf{s},\mathbf{a},r \in \mathbf{h}} \Phi(\mathbf{z} \mid \mathbf{s}, \mathbf{a}, r),$$

| Hyperparameter | Value |
|---|---|
| RL batch size | 256 |
| encoder batch size | 64 |
| meta batch size | 4 |
| Q-network hidden sizes | $[300, 300, 300]$ |
| policy network hidden sizes | $[300, 300, 300]$ |
| decoder network hidden sizes | $[64, 64]$ |
| encoder network hidden sizes | $[200, 200, 200]$ |
| $\mathbf{z}$ dimensionality ($d_z$) | 5 |
| hidden activation (all networks) | ReLU |
| Q-network, encoder, and decoder output activation | identity |
| policy output activation | tanh |
| discount factor $\gamma$ | 0.99 |
| target network soft target $\tau$ | 0.005 |
| policy, Q-network, encoder, and decoder learning rate | $3 \times 10^{-4}$ |
| policy, Q-network, encoder, and decoder optimizer | Adam |
| # of gradient steps per environment transition | 4 |

**Table E.4:** SMAC Hyperparameters for Self-Supervised Phase

| Hyperparameter | Cheetah | Ant | Sawyer |
|---|---|---|---|
| max trajectory length | 200 | 200 | 50 |
| AWR $\beta$ | 100 | 100 | 0.3 |
| reward scale | 5 | 5 | 1 |
| # of training tasks | 100 | 100 | 50 |
| # of test tasks | 30 | 20 | 10 |
| # offline transitions per task | 1600 | 1600 | 3750 |
| $\lambda_{\text{pearl}}$ | 1 | 1 | 0 |

**Table E.5:** Environment Specific SMAC Hyperparameters

where each factor is a multi-variate Gaussian over $\mathbb{R}^{d_z}$ with learned mean and diagonal variance. In other words,

$$q_\phi(\mathbf{z} \mid \mathbf{s}, \mathbf{a}, r) = \mathcal{N}(\mu_\phi(\mathbf{s}, \mathbf{a}, r), \sigma_\phi(\mathbf{s}, \mathbf{a}, r)).$$

The mean and standard deviation is the output of a single MLP network with output dimensionality $2 \times d_z$. The output of the MLP network is split into two halves. The first half is the mean and the second half is passed through the softplus activation to get the standard deviation.

**Self-supervised actor update.** The parameter $\lambda_{\text{pearl}}$ controls the actor loss during the self-supervised phase, which is

$$\mathcal{L}_{\text{actor}}^{\text{self-supervised}}(\theta) = \mathcal{L}_{\text{actor}}(\theta) + \lambda_{\text{pearl}} \cdot \mathcal{L}_{\text{actor}}^{\text{PEARL}}(\theta),$$

where $\mathcal{L}_{\text{actor}}^{\text{PEARL}}$ is the actor loss from PEARL [176]. For reference, the PEARL actor loss is

$$\mathcal{L}_{\text{actor}}^{\text{PEARL}}(\theta) = \mathbb{E}_{\mathbf{s} \sim \mathcal{D}_i, \mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{h})} \left[ D_{\text{KL}} \left( \pi(\mathbf{a} \mid \mathbf{s}, \mathbf{z}) \middle|\middle| \frac{\exp Q_w(\mathbf{s}, \mathbf{a}, \mathbf{z})}{Z(\mathbf{s})} \right) \right].$$

When the parameter $\lambda_{\text{pearl}}$ is zero, the actor update is equivalent to the actor update in AWAC [157].