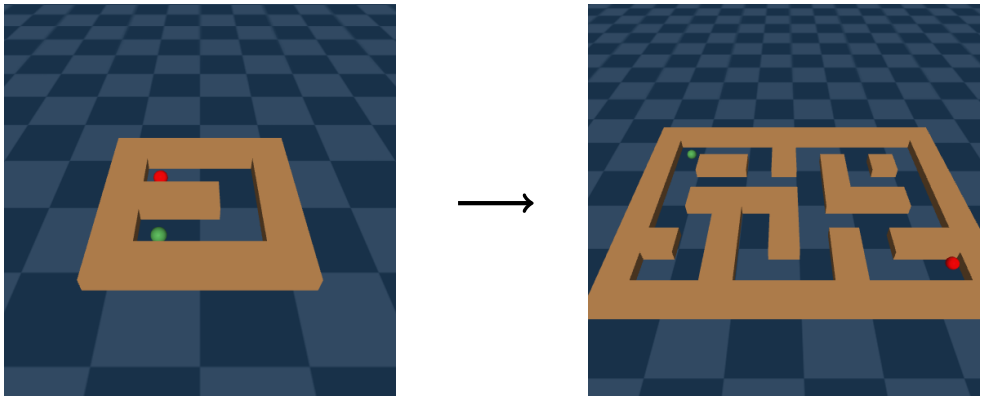
Simple-to-hard generalization

Authors. Adithya Bhaskar (adithyab@princeton.edu), Laura Hwa (yh2673@princeton.edu), and Katie Heller (kh3329@princeton.edu)

Link to GitHub Repo. https://github.com/cos435-simpletohard/simpletohard

1 Introduction



(a) Train in "simple" environment (b) Evaluate in "hard" environment

Figure 1: Simple-to-hard generalization in RL

This report studies the problem of *simple-to-hard generalization* in Reinforcement Learning (RL): if one trains a policy on an environment, does the policy perform well on a similar, but more complex, version (Figure 1)? There are several reasons generalizing policies are desirable, chief among them the fact that we would like our policies to work well when deployed under varying and unexpected conditions. Indeed, it is infeasible in most cases to simulate exactly the real environment during the training process.

Interest in policy generalization [1] has existed since the early days of RL, but has accelerated in recent years as techniques have improved. Some common ways to study generalization include perturbing the dynamics [7] or reward [9]. Cobbe et al. [2]'s work is perhaps the closest to ours: they train policies on video game levels and evaluate them on other procedurally generated levels. Recently, Myers et al. [6] studied the notion of *horizon generalization* under some notion of distance.

This report studies specifically the generalization of *goal-conditioned* policies [4]: policies that are conditioned on, and try to achieve, a specified goal state. In principle, goal-conditioned policies are uniquely positioned to generalize to similar environments, as they fit environment dynamics, not explicit rewards. A reward is only specified by fixing a certain goal. So, we reason that if the dynamics transfer between the environments, the policies would as well. We examine this hypothesis with the SGCRL algorithm ("A single goal is all you need") [5], a modification of the contrastive reinforcement learning (CRL) algorithm [3] (more details in Section 2).

Our project follows a three part structure:

1. First, we reproduce the results of Liu et al. [5] on *Sawyer Box*, conditioning on the coordinates provided by the environment.

2. Then, we shift to *Point Maze*, which has a clearer notion of "simple" and "hard" forms of the same environment dynamics. To unify the representation of various mazes, we condition on scene renders rather than coordinates (Section 2).
3. We train policies on the U, medium, and hard versions of the maze. Upon evaluation, we find that none of the policies generalize to the other mazes (Section 3).

Our results indicate that achieving simple-to-hard generalization is nontrivial; we suggest that future work try addressing this problem. However, we also caution that our results were bottlenecked by fewer training steps and a single choice of image embeddings. It is possible that when differently embedded, the environments might lend themselves to better-generalizing policies. We discuss these issues further in Section 4.

2 Methods

The SGCRL algorithm We begin with a brief overview of the SGCRL algorithm. SGCRL operates within the actor-critic framework, where the actor is a policy conditioned on both the current state s and a goal state g. The critic consists of two networks $\phi(s,a), \psi(g)$ that embed state-action pairs and goals, respectively. The three defining characteristics of SGCRL are:

1. The critic optimizes the objective

$$\max_{\phi,\psi} \mathop{\mathbb{E}}_{\substack{(s,a)\sim p(s,a), g^{(1)}\sim\rho(g|s,a) \\ g^{(2:N)}\sim\rho(g)}} \left[\underbrace{\log\left(\frac{\exp\left(\phi(s,a)^T\psi(g^{(1)})\right)}{\sum_{i=1}^{N}\phi(s,a)^T\psi(g^{(i)})} \right)}_{\text{infoNCE}} - 0.01 \cdot \underbrace{\log\left(\sum_{i=1}^{N}\exp\left(\phi(s,a)^T\psi(g^{(i)})\right)\right)^2}_{\text{regularization}} \right] \tag{1}$$

   This objective is to be interpreted as follows: we sample state-action pairs (s,a) from the replay buffer, and actual future states $g^{(1)}$ that were reached by the corresponding trajectory after $\Delta \sim \text{Geom}(1-\gamma)$ steps. We also sample $N-1$ "negatives" $g^{(2:N)}$ from the marginal distribution of future states. Then, the critic gets better at identifying the correct future state. The regularization term prevents the dot products from growing without bound. Then, $\psi(s,a)^T\psi(g)$ is a surrogate for the probability that the action a in state s leads eventually to the goal g[1].
2. The actor π simply wants to learn how to reach any specified goal. To this end, we sample a state s and goal g from the replay buffer, and optimize

$$\max_{\pi} \mathop{\mathbb{E}}_{\substack{s,g\sim p(s,g) \\ a\sim\pi(s,g)}} \left[\phi(s,a)^T\psi(g) + \alpha\mathbf{H}(\pi(\cdot|s,g))\right] \tag{2}$$

   The second term is a maximum-entropy term to encourage exploration. In our experiments, we parameterize $\log\alpha$ by initializing it to 0 and updating it as *Lagrange parameter* (i.e., trying to minimize the objective instead of maximizing it).
3. The replay buffer is continuously filled by sampling trajectories from the actor. These trajectories are always conditioned on the true goal $g^\star$ (this is where the name "single goal" comes from).

How shall we embed the state? In the Sawyer Box environment, both the state and the action are embedded as vectors. The former consists of the coordinates of the robotic arm, the lid itself, and the orientation of the lid. The action is a four-dimensional vector representing the three axes of movement and the grip/release motions of the claw.

Point Maze provides us with the coordinates of the ball and the goal, along with the former's velocity. However, this representation is at odds with our goal of testing simple-to-hard generalization. This discrepancy arises because these coordinates do not encode any information about the topology of the maze; all such information is only present in the

[1]Technically, it represents the *discounted state occupancy measure of the goal g.*

learned weights of the actor and critic networks. It would be unreasonable to expect our policy to generalize to any other maze, since it has memorized the maze it was trained on.

To fix this issue, we can ensure that our state embeddings also convey the topology of the entire maze. To do so, we choose to render the entire maze as a 256×256 RGB image[2]. The memory requirement of these renders quickly grows unreasonable, so we embed every render using a CLIP [8] encoder into a 512d vector, and throw away the renders themselves. To convey the velocity of the ball, we embed three frames at timesteps $(t-10, t-5, t)$ where t is the current timestep (values ≤ 0 are rounded up to 1), leading to a 1536-dimensional input. An obvious limitation crops up here: our embedder has not been trained on the distribution of Point Maze renders. The generality of the CLIP encoder should assuage this issue, but future work can try training the image encoder further before RL.

Network architectures and hyperparameters All of our networks (the actor, state-action encoder, and goal encoder) are 3-layer MLPs with all hidden sizes set to 256. Our state-action and goal embeddings are 64-dimensional each. We provide a complete list of hyperparameters for the Sawyer Box and Point Maze environments in Appendix A.

Evaluation On Sawyer Box, we do not conduct any extrinsic evaluation. The environment provides us with (soft) rewards, and a 0/1 value indicating success. While we do not train on the reward, we plot it throughout training along with the actor/critic losses. We also observe a curriculum of skills emerge as in Liu et al. [5]; we present some renders to convey this curriculum.

On Point Maze, we report actor and critic losses during training. The environment provides soft rewards, but we found that these were 0 except when the ball was sufficiently close to the goal. At close proximity, the reward was generally a small positive value inversely correlated with distance. We plot this. We also evaluate each of the {U, Medium, Large} mazes with the policy trained each of the {U, Medium, Large} mazes with 100 random seeds. The seeds do not affect the initial position of the ball and the goal, only the actions sampled from the policy's output distribution. We report the success rate of each {policy, environment} pair in a tabular format (Table 1).

3 Results

3.1 Sawyer Box, training with coordinates

Loss and reward curves We plot the actor and critic losses throughout training in Figure 2. We see that neither overpowers the other, and each challenges the other networks to learn better. The reward, plotted in Figure 3 stays low until around 35,000 steps—at which point it starts increasing. The actor observes its first success at 42,000 steps (marked with a star in Figure 3), though it takes around 5,000 more steps for it to consistently succeed.

Emergence of skills The SGCRL paper reports the gradual emergence of skills throughout training, long before the reward spikes [5]. We also observe the same phenomenon, which we depict in Figure 4: the arm first tries reaching for the goal (8,250 steps), then nudges the lid (12,570 steps), then drags it around (31,020 steps), lifts it (50,720 steps), and then finally places it on the box (54,000 steps). We were surprised that the first success occurred before the last two; on inspection, we found that the environment considers a state a success even if the lid is not perfectly oriented. This likely explains why our first success came before skills d) and e) emerged.

3.2 Point Maze, training with pixels

Can we learn good policies? We plot the rewards obtained by the policies on the U, Medium, and Large mazes in Figure 5, and the actor and critic losses in Figure 6. We find

[2]The default camera position does not cover the entirety of the Large maze; we had to increase its elevation and field of view.

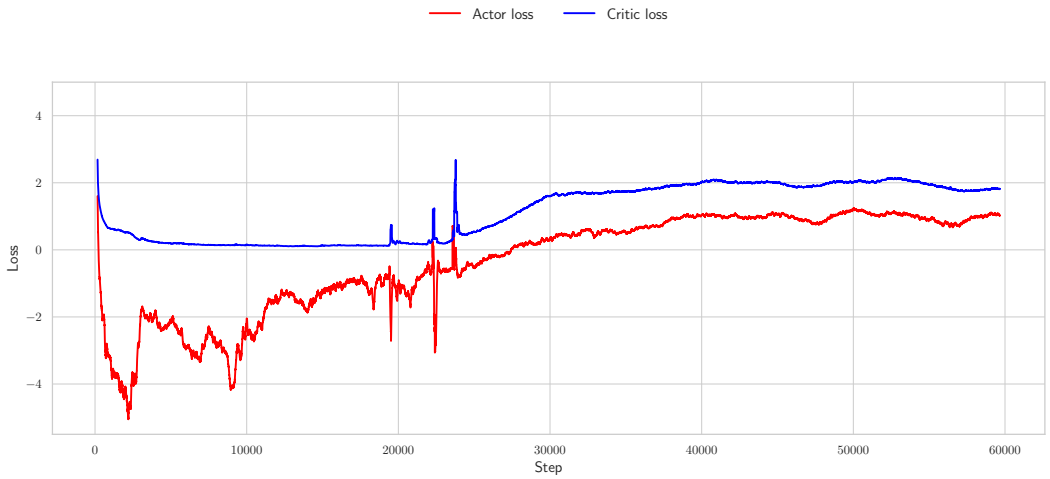


Figure 2: Actor and critic losses for Sawyer Box (coordinates).

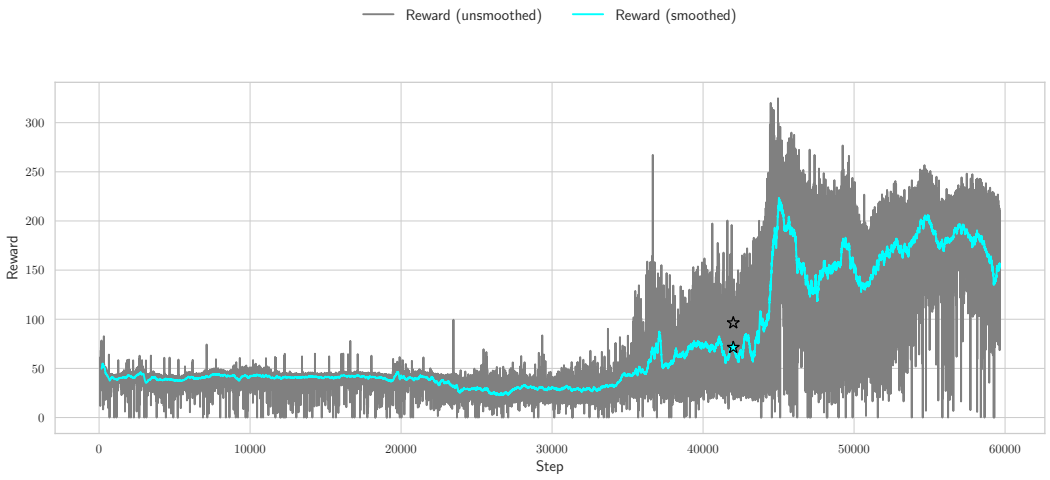
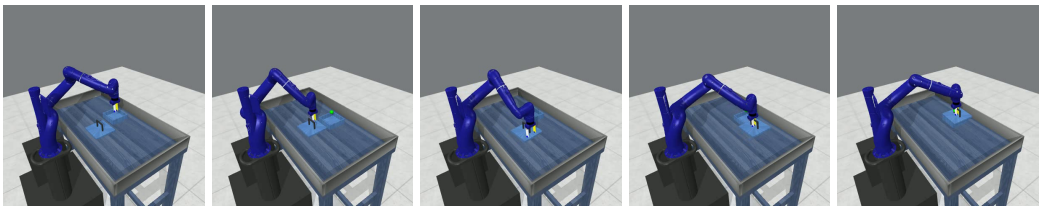
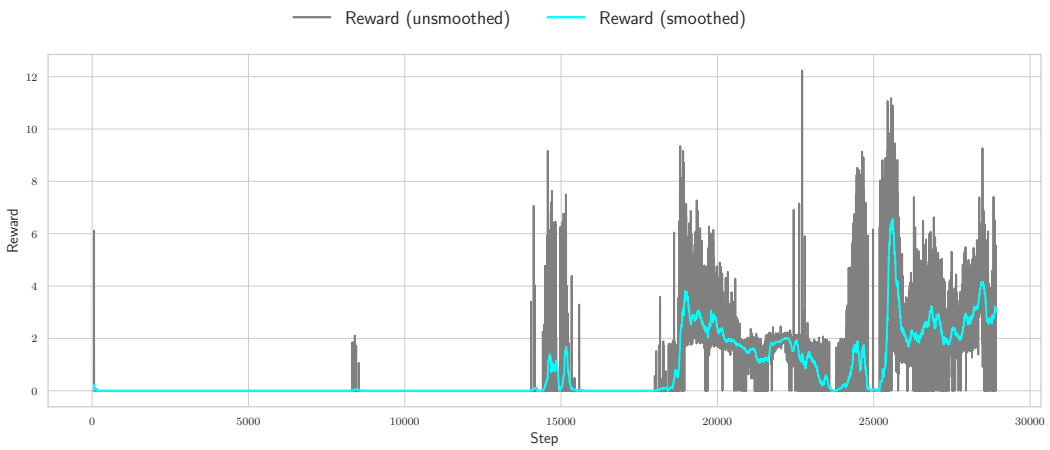


Figure 3: Reward v/s training steps for Sawyer Box (coordinates). We observe the first success at 42,000 steps.

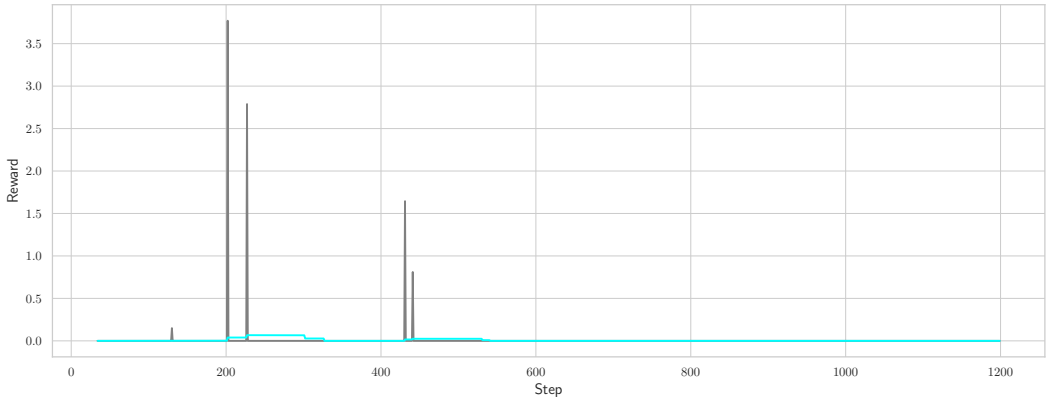


(a) Reaching (b) Nudging (c) Dragging (d) Lifting (e) Placing

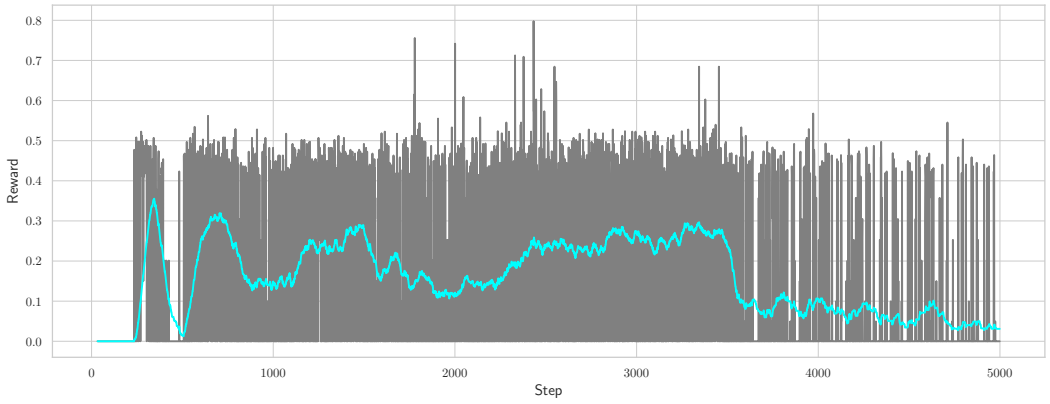
Figure 4: A curriculum of skills emerges over the course of training (Sawyer Box).



(a) U Maze

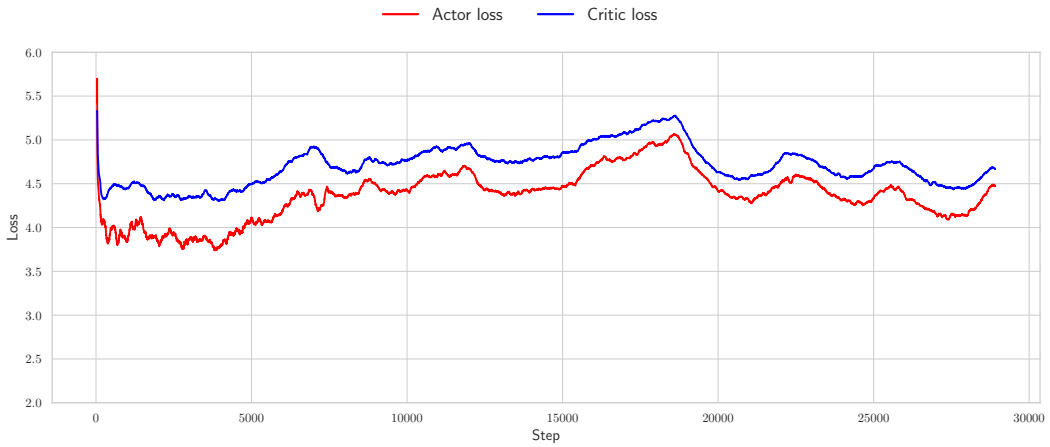


(b) Medium Maze

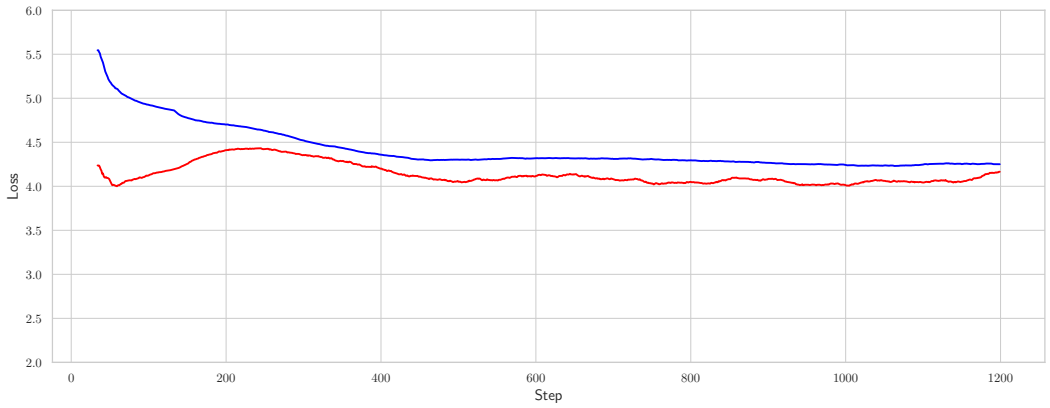


(c) Large Maze

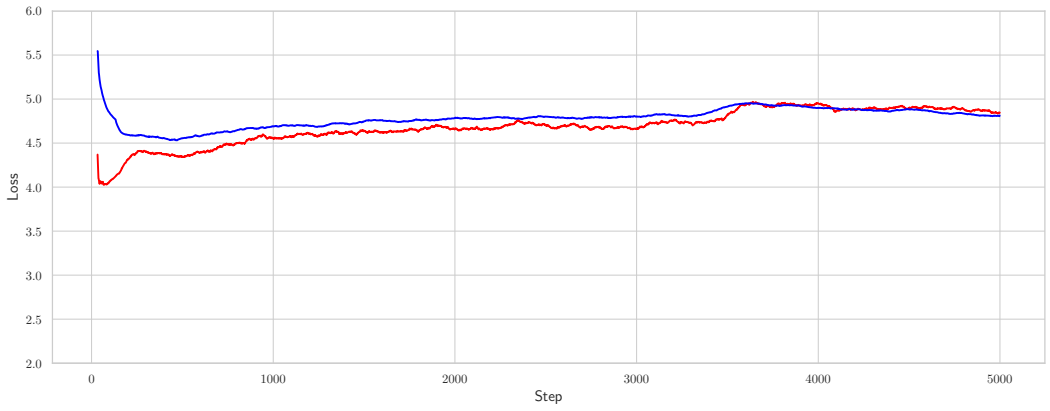
Figure 5: The reward on Point Maze over the course of training (trained on embedded renders).



(a) U Maze



(b) Medium Maze



(c) Large Maze

Figure 6: The actor and critic losses on Point Maze over the course of training (trained on embedded renders).

that the CLIP encoders endow our policies with a very strong inductive bias. Our policies succeed earlier than 1,000 steps on both Medium and Large maze. We are unsure why the reward graph is sparse for the Medium maze. One possible explanation is that the ball rolls right next to the goal but then keeps ramming into the wall directly below it—these positions are nonetheless marked as successes by the environment. Once again, the actor and critic losses are stable throughout training. We show example trajectories drawn from the learned policies in Appendix B.

Table 1: The policies trained on one maze do not generalize to other mazes.

	Evaluated on		
Trained on	U	Medium	Hard
U	✓ 100/100	✗ 0/100	✗ 0/100
Medium	✗ 0/100	✓ 100/100	✗ 0/100
Hard	✗ 0/100	✗ 0/100	✓ 100/100

Do the policies generalize? Having trained the policies on each maze, we evaluate each policy on each maze, as described in Section 2. We find that none of the policies generalize to the other mazes. The ball simply runs into an adjacent wall in all cases, not demonstrating any significant progress.

4 Discussion and Limitations

Training a generalizing goal-conditioned policy is non-trivial Our experiments show that although pixels offer a unified representation of mazes, the learned policies overfit to their specific environments. We believe that it is possible to learn goal-conditioned policies that generalize and encourage future work to tackle this problem.

Limitations of the SGCRL algorithm: We found the SGCRL algorithm to be very sensitive to the exact choices of the hyperparameters. Slight variations in the learning rate or batch size marked the difference between success and failure. Although we do not have quantitative metrics to represent this point, we believe that future work should also try to make goal-conditioned algorithms easier to train.

Limitations of our implementation Our implementation of SGCRL is much slower than the official one[3], averaging around 1,500 training steps per hour. We attribute this to several reasons:

1. We use PyTorch instead of JAX, which makes our code slower.
2. We rely on renders of the scene instead of coordinates, which introduces a significant latency to every environment step.
3. Our implementation of the replay buffer is not sufficiently parallelized.

Limitations of our problem setting Our conclusions should be taken with the caveat that the CLIP encoder has not been trained on Point Maze renders. While the CLIP encoder has been pretrained on a diverse array of images and was sufficient to succeed in the specific environments, a better image embedding might allow trained policies to generalize. Similarly, experimental takeaways might also differ for other environments. Further training the embedder on scene renders may also be a helpful next step, which we leave to future work.

Acknowledgements

We thank the COS 435 staff (Ben and the TAs) for their help in ideation and debugging.

[3] https://github.com/graliuce/sgcrl

References

[1] Boyan, J. A. and Moore, A. W. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 369–376.

[2] Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. (2019). Quantifying generalization in reinforcement learning. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1282–1289. PMLR.

[3] Eysenbach, B., Zhang, T., Levine, S., and Salakhutdinov, R. (2022). Contrastive learning as goal-conditioned reinforcement learning. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

[4] Kaelbling, L. P. (1993). Learning to achieve goals. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2, pages 1094–1098. Morgan Kaufmann.

[5] Liu, G., Tang, M., and Eysenbach, B. (2025). A single goal is all you need: Skills and exploration emerge from contrastive RL without rewards, demonstrations, or subgoals. In *The Thirteenth International Conference on Learning Representations*.

[6] Myers, V., Ji, C., and Eysenbach, B. (2025). Horizon generalization in reinforcement learning. In *The Thirteenth International Conference on Learning Representations*.

[7] Packer, C., Gao, K., Kos, J., Krahenbuhl, P., Koltun, V., and Song, D. (2019). Assessing generalization in deep reinforcement learning.

[8] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. (2021). Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *PMLR*, pages 8748–8763.

[9] Zhang, C., Vinyals, O., Munos, R., and Bengio, S. (2018). A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*.

A Hyperparameters

We provide the hyperparameters we used for Sawyer Box and Point Maze in Tables 2 and 3, respectively.

Table 2: Hyperparameters for Sawyer Box (Coordinates)

Hyperparameter	**Value**
Discount Factor (γ)	0.99
Actor Learning Rate	3×10^{-4}
Critic Learning Rate	3×10^{-4}
Replay Buffer Size	10^6
Num. Transitions Before Training	10^4
Max Episode Length	150
Total Training Steps	10^6
Actor Batch Size	16,384
Actor Micro-Batch Size	256
Critic Batch Size	1,024
Critic Micro-Batch Size	1,024
Max Log Std	5.0
Min Log Std	-5.0
Max Entropy Coefficient	0.0
Network Architectures	
Actor	Input: $2 \times d_{obs}$ (State + Goal) FC($2d_{obs}$, 256) $\rightarrow$ ReLU $\rightarrow$ FC(256, 256) $\rightarrow$ ReLU $\rightarrow$ FC(256, d_{act})
State-Action Encoder	Input: $d_{obs} + d_{act}$ (State + Action) FC($d_{obs} + d_{act}$, 256) $\rightarrow$ ReLU $\rightarrow$ FC(256, 256) $\rightarrow$ ReLU $\rightarrow$ FC(256, 64)
Goal Encoder	Input: d_{obs} (Goal) FC(d_{obs}, 256) $\rightarrow$ ReLU $\rightarrow$ FC(256, 256) $\rightarrow$ ReLU $\rightarrow$ FC(256, 64)

Table 3: Hyperparameters for Point Maze (Rendered)

Hyperparameter	**Value**
Discount Factor	0.99
Actor Learning Rate	3×10^{-4}
Critic Learning Rate	3×10^{-4}
$\log \alpha$ Learning Rate	3×10^{-4}
Replay Buffer Size	10^6
Num. Transitions Before Training	10^4
Max Episode Length	150
Total Training Steps	10^6
Actor Batch Size	16,384
Actor Micro-Batch Size	256
Critic Batch Size	1,024
Critic Micro-Batch Size	1,024
Max Log Std	5.0
Min Log Std	-5.0
Target Entropy	0
Image Embedding Dimension (d_{img})	512
Num. frames (n_{frames})	3
Frame Skip (δ_{frame})	5
CLIP Checkpoint	openai/clip-vit-base-patch32
Network Architectures	
Actor	Input: $n_{frames} \times d_{img}$ (State) + d_{img} (Goal) FC($n_{frames}d_{img} + d_{img}$, 256) $\rightarrow$ ReLU $\rightarrow$ FC(256, 256) $\rightarrow$ ReLU $\rightarrow$ FC(256, d_{act})
State-Action Encoder	Input: $n_{frames} \times d_{img}$ (State) + d_{act} (Action) FC($n_{frames}d_{img} + d_{act}$, 256) $\rightarrow$ ReLU $\rightarrow$ FC(256, 256) $\rightarrow$ ReLU $\rightarrow$ FC(256, 64)
Goal Encoder	Input: d_{img} (Goal) FC(d_{img}, 256) $\rightarrow$ ReLU $\rightarrow$ FC(256, 256) $\rightarrow$ ReLU $\rightarrow$ FC(256, 64)

B Example trajectories from successful runs

In this appendix, we show example trajectories from successful runs on Point Maze. We show trajectories for the U, medium, and large mazes in Figures 7, 8, and 9, respectively.

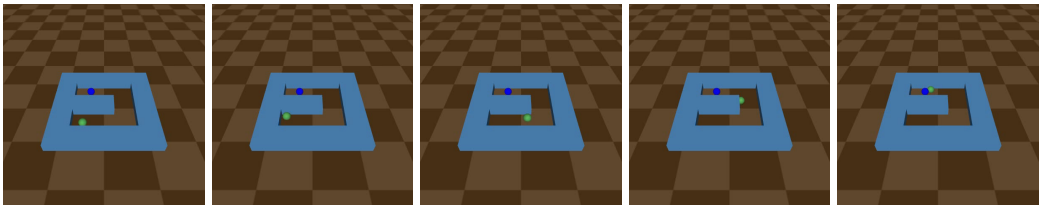


Figure 7: A trajectory drawn from the policy learned on the U maze.

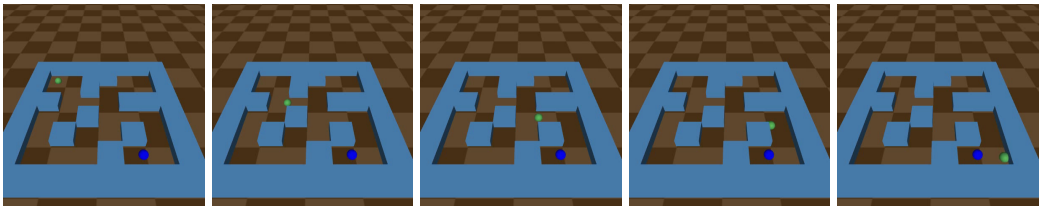


Figure 8: A trajectory drawn from the policy learned on the medium maze.

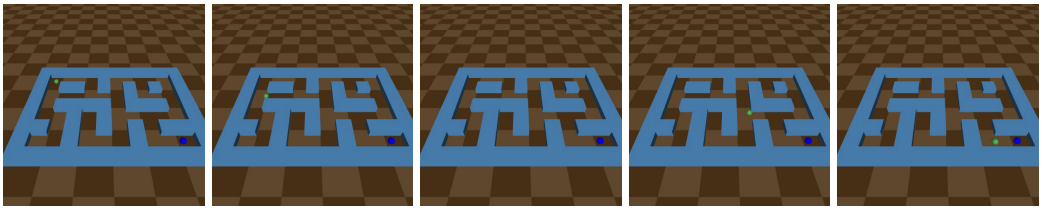


Figure 9: A trajectory drawn from the policy learned on the large maze.