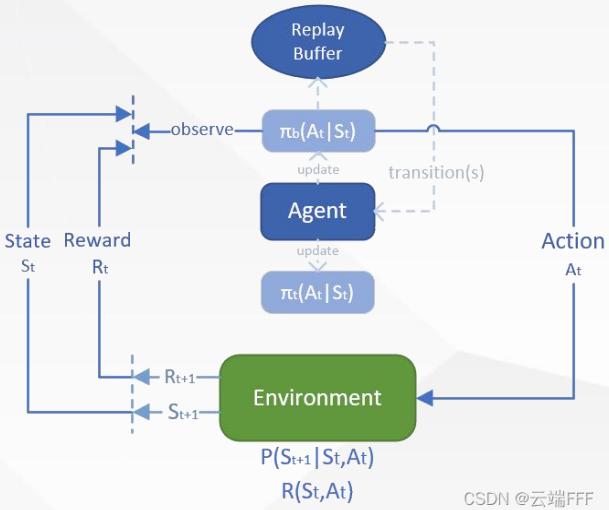
Published as a conference paper at ICLR 2022

RvS: What is Essential for Offline RL via Supervised Learning?

Scott Emmons[1], **Benjamin Eysenbach**[2], **Ilya Kostrikov**[1], **Sergey Levine**[1]
[1]UC Berkeley, [2]Carnegie Mellon University
emmons@berkeley.edu

Offline Reinforcement Learning

Online Reinforcement Learning

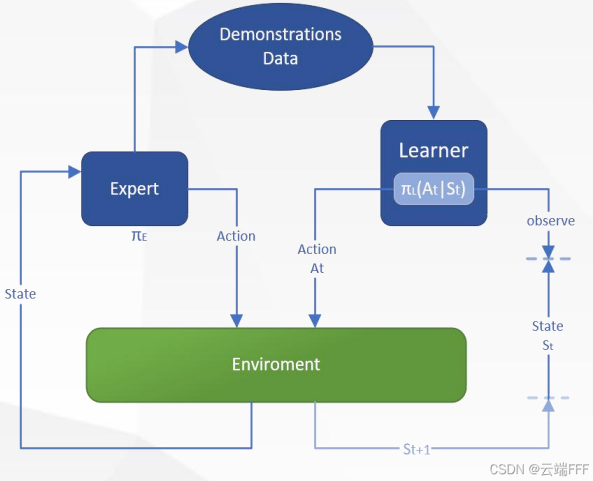


- 从环境交互中学习
- 需要设计 reward function

交互成本高

模拟器成本高

Imitation Learning

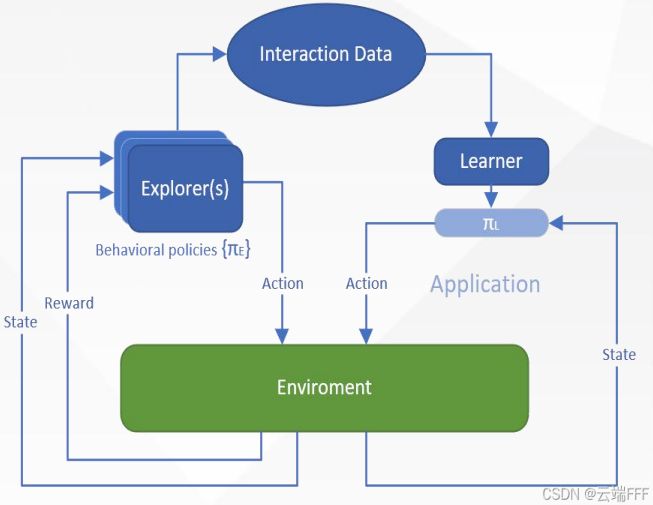


- 从专家示范中学习
- 无需 reward function
- 允许环境交互

专家示范成本高

有时需要可交互环境

Offline Reinforcement Learning



- 从混合轨迹样本中学习
- 存在 reward signal
- 禁止环境交互

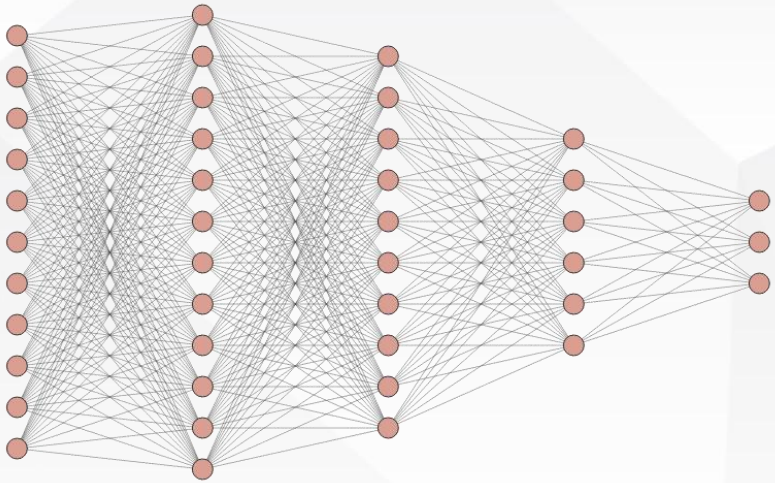
几乎无额外成本

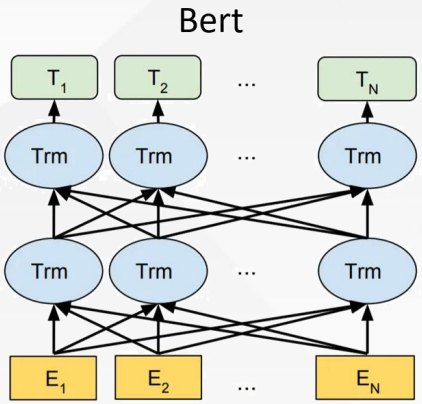
Offline Reinforcement Learning

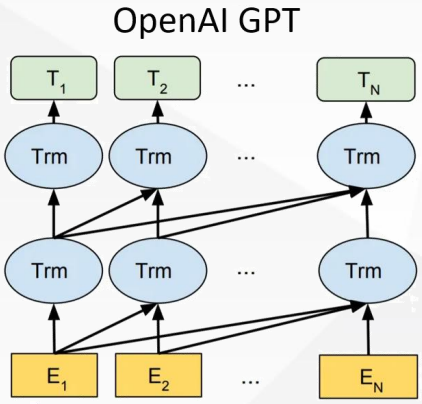
- 现有方法分类

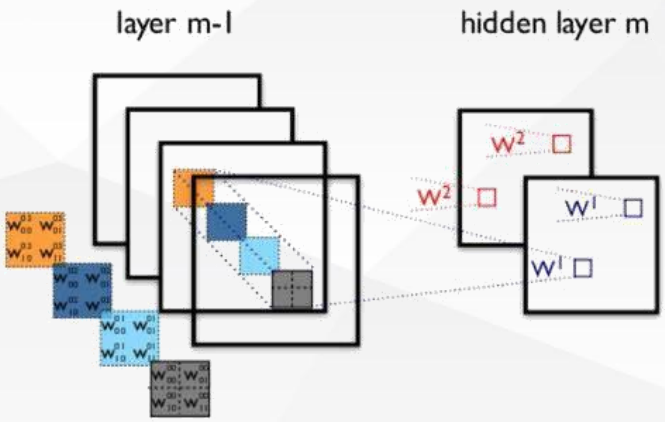
- 直接上 Off-policy 方法不可行：Extrapolation error
  - Offline Dataset 覆盖的 (s,a) 区域价值估计不好
  - 测试时遇到的 (s,a) 分布有 Mismatch 问题

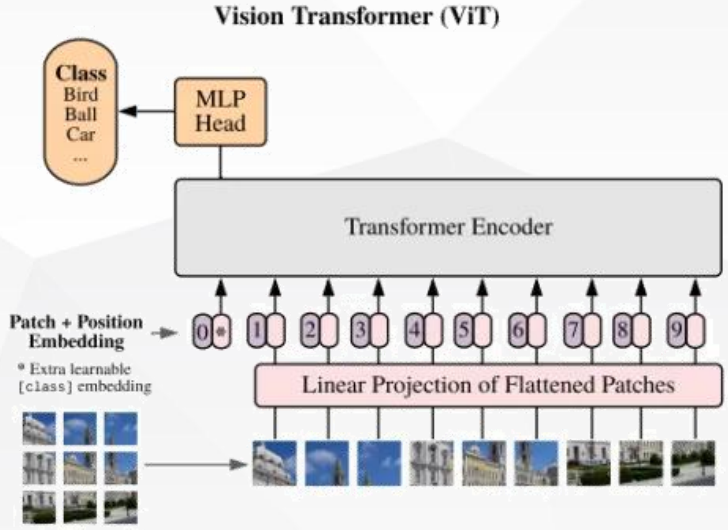
Capacity perspective on SL







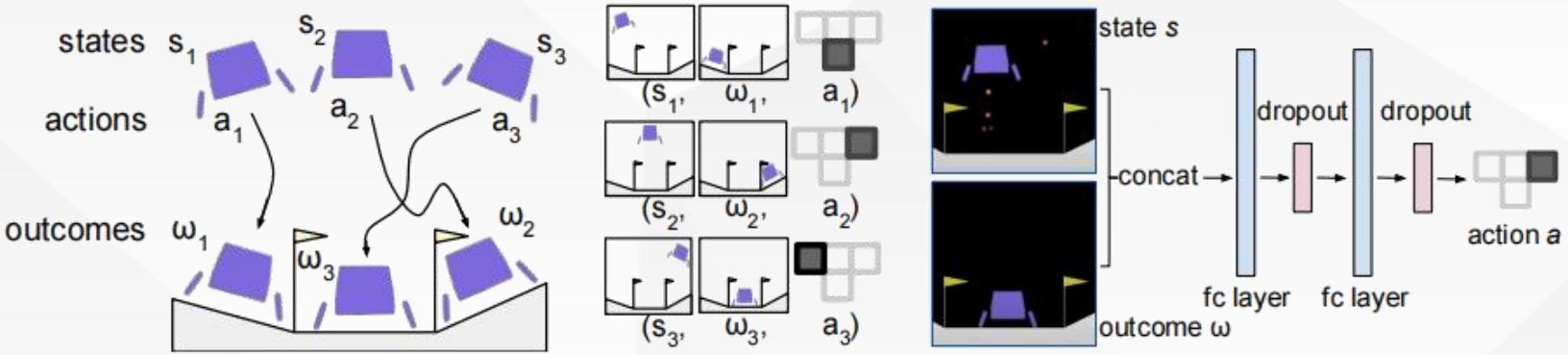




RvS Learning

- What elements are essential for effective RvS learning?
  - Sample FNN with proper capacity and regularization is good enough
- What are the limits of RvS learning?
  - Choosing conditioning variable can be hard
  - No reliable metric for automatically hyperparameters tuning
- Does it scale to settings with few near-optimal trajectories?
  - yes

RvS Learning



(a) replay buffer (b) training dataset (c) network architecture

Figure 2: (a) As input, RvS takes a precollected replay buffer of experience. An outcome ω can be an arbitrary function of the trajectory, such as future states or rewards. (b) RvS uses hindsight relabeling of the replay buffer to construct a training dataset. The observed actions act as demonstrations for the observed outcomes. (c) Our implementation of RvS uses an MLP with two fully connected (fc) layers to predict actions. At test time, we can condition on arbitrary outcomes.

RvS Learning

goal-conditioned (RvS-G) $f(\omega \mid \tau_{t:H}) = \text{Unif}(s_{t+1}, s_{t+2}, \cdots, s_H)$

reward-conditioned (RvS-R) $f(\omega \mid \tau_{t:H}) = \mathbb{1}(\omega = \frac{1}{H-t+1} \sum_{t'=t}^{H} r(s_{t'}, a_{t'}))$

For all trajectories: For all timesteps in that trajectory: For all achieved outcomes:

$$\max_\theta \sum_{\tau \in \mathcal{D}} \sum_{1 \le t \le |\tau|} \mathbb{E}_{\omega \sim f(\omega \mid \tau_{t:H})} [\log \pi_\theta(a_t \mid s_t, \omega)].$$

Algorithm 1 RvS-Learning

1: **Input**: Dataset of trajectories, $\mathcal{D} = \{\tau\}$
2: Initialize policy $\pi_\theta(a \mid s, \omega)$.
3: **while** not converged **do**
4: Randomly sample trajectories: $\tau \sim \mathcal{D}$.
5: Sample time index for each trajetory, $t \sim [1, H]$, and sample a corresponding outcome: $\omega \sim f(\omega \mid \tau_{t:H})$.
6: Compute loss: $\mathcal{L}(\theta) \leftarrow \sum_{(s_t, a_t, \omega)} \log \pi_\theta(a_t \mid s_t, \omega)$
7: Update policy parameters: $\theta \leftarrow \theta + \eta \nabla_\theta \mathcal{L}(\theta)$
8: **end while**
9: **return** Conditional policy $\pi_\theta(a \mid s, \omega)$

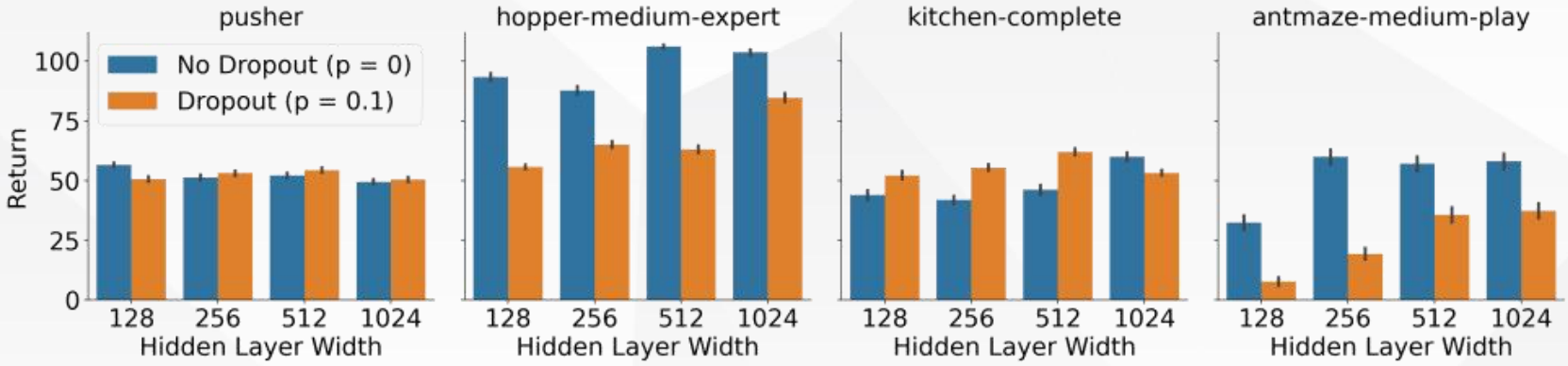
Experiment

- Exp Envs
  - GCSL: A suite of goal-conditioned environments. Random policy data
  - Gym Locomotion: Requires learning from mixed and suboptimal data, reward-maximization tasks. D4RL data
  - Franka Kitchen: Requires composing multi-step behaviors from component skills. D4RL data
  - AntMaze: Test an agent's ability to learn temporal compositionality by combining subtrajectories of different demonstrations. D4RL data

Overall performance

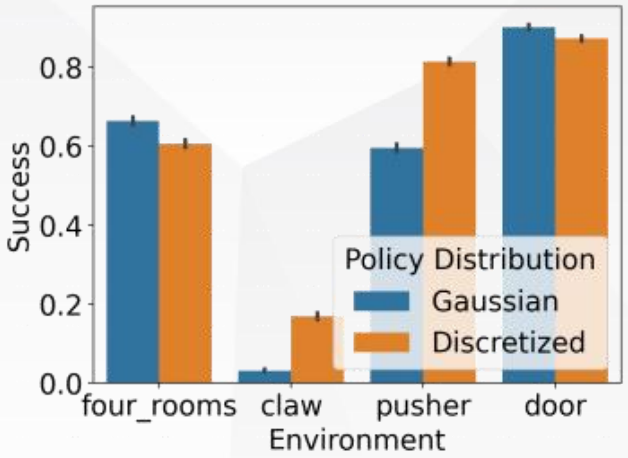
Suite	Environment	BC	Filt. BC	TD3+BC	Onestep	CQL-r	CQL-p	DT	RvS-R	RvS-G	GCSL
D4RL AntMaze	umaze-v2	54.6	60.0	**78.6**	64.3	44.8	**74.0**	65.6	64.4	65.4	
	umaze-diverse-v2	45.6	46.5	71.4	60.7	23.4	**84.0**	51.2	70.1	60.9	
	medium-play-v2	0.0	42.1	10.6	0.3	0.0	**61.2**	1.0	4.5	**58.1**	
	medium-diverse-v2	0.0	37.2	3.0	0.0	0.0	53.7	0.6	7.7	**67.3**	
	large-play-v2	0.0	**28.0**	0.2	0.0	0.0	15.8	0.0	3.5	**32.4**	
	large-diverse-v2	0.0	**34.3**	0.0	0.0	0.0	14.9	0.2	3.7	**36.9**	
	antmze-v2 average	16.7	41.4	27.3*	20.9*	11.4	**50.6***	19.8	25.6	**53.5**	
D4RL Gym	halfcheetah-random-v2	2.3	2.0	11.0	6.9	**18.6**		2.2	3.9		
	hopper-random-v2	**4.8**	4.1	**8.5**	**7.8**	**9.3**		**7.5**	**7.7**		
	walker2d-random-v2	**1.7**	**1.7**	**1.6**	**6.1**	**2.5**		**2.0**	-0.2		
	random-v2 average	2.9	2.6	**7.0**	**6.9**	**10.1**		3.9	3.8		
	halfcheetah-medium-replay-v2	36.6	40.6	**44.6**	**42.4**	**47.3**		36.6	38.0		
	hopper-medium-replay-v2	18.1	75.9	60.9	71.0	**97.8**		82.7	73.5		
	walker2d-medium-replay-v2	26.0	62.5	**81.8**	71.6	**86.1**		66.6	60.6		
	medium-replay-v2 average	26.9	59.7	62.4	61.7	**77.1**		62.0	57.4		
	halfcheetah-medium-v2	42.6	42.5	48.3	**55.6**	49.1		42.6	41.6		
	hopper-medium-v2	52.9	56.9	59.3	**83.3**	64.6		67.6	60.2		
	walker2d-medium-v2	75.3	75.0	**83.7**	**85.6**	**82.9**		74.0	71.7		
	medium-v2 average	56.9	58.1	63.8	**74.8**	65.5		61.4	57.8		
	halfcheetah-medium-expert-v2	55.2	**92.9**	**90.7**	**93.5**	85.8		86.8	**92.2**		
	hopper-medium-expert-v2	52.5	**110.9**	98.0	102.1	102.0		**107.6**	101.7		
	walker2d-medium-expert-v2	**107.5**	**109.0**	**110.1**	**110.9**	**109.5**		**108.1**	**106.0**		
	medium-expert-v2 average	71.7	**104.3**	**99.6**	**102.2**	99.1		**100.8**	**100.0**		
	gym-v2 average	39.6	56.2	**58.2**	**61.4**	**63.0**		57.0	54.7		
D4RL Kitchen	kitchen-complete-v0	**65.0**	4.0			11.8	43.8		1.5	50.2	
	kitchen-mixed-v0	51.5	40.0			24.2	51.0		1.1	**60.3**	
	kitchen-partial-v0	38.0	**66.0**			20.8	49.8		0.5	51.4	
	kitchen-v0 average	**51.5**	36.7			18.9	48.2		1.0	**54.0**	
GCSL	claw									14.8	**28.0**
	door									**95.8**	28.0
	four rooms									63.0	**81.0**
	lunar									54.7	**70.0**
	pusher									**81.8**	**83.0**
	gcsl average									**62.0**	**58.0**

Capacity and regularization



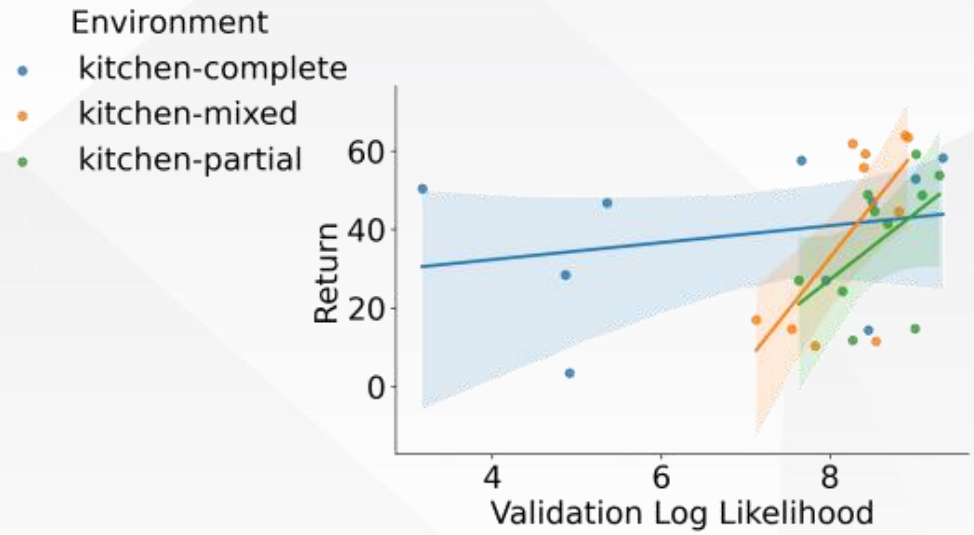
- The best-performing architectures are generally larger than the architectures used in standard online RL and IL
- Dropout improves performance in some tasks and worsens performance in other tasks, sometimes has no effect as well, which maybe depends on the relative complexity of architectures

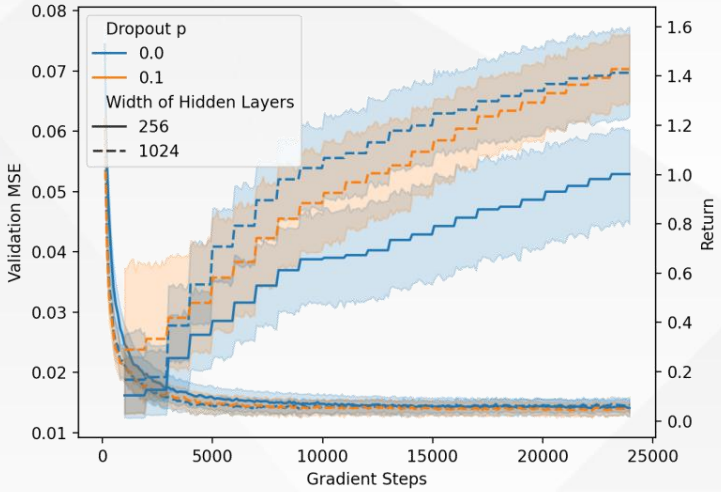
Output distributions



- The form of the policy's output distribution also influences the policy's capacity, increasing the policy's model capacity can improve performance
- Upweighting better data as done by previous methods maybe reduced data complexity so that Gaussian policy can performs well

Tuning with validation loss





- Validation set error does correlate with performance, but the strength of this correlation varies significantly, can't be a reliable approach for hyperparameter tuning

A recipe for practitioners. We suggest the following process for online hyperparameter tuning: incrementally increase network width until performance saturates, and then try adding a bit of dropout regularization (e.g., dropout $p = 0.1$). If validation loss indicates that under/overfitting is an issue, one can also try increasing/decreasing the batch size, respectively.

Subtrajectory stitching

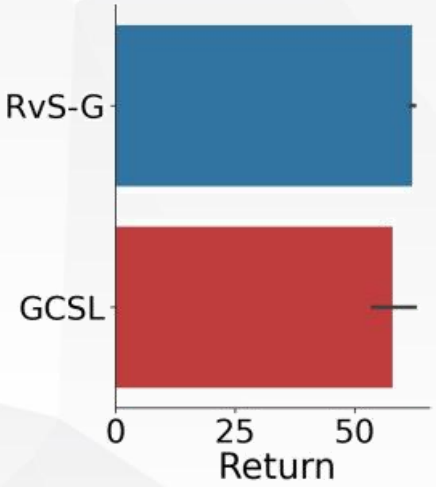
Suite	Environment	BC	Filt. BC	TD3+BC	Onestep	CQL-r	CQL-p	DT	RvS-R	RvS-G	GCSL
D4RL AntMaze	umaze-v2	54.6	60.0	**78.6**	64.3	44.8	**74.0**	65.6	64.4	65.4	
	umaze-diverse-v2	45.6	46.5	71.4	60.7	23.4	**84.0**	51.2	70.1	60.9	
	medium-play-v2	0.0	42.1	10.6	0.3	0.0	**61.2**	1.0	4.5	**58.1**	
	medium-diverse-v2	0.0	37.2	3.0	0.0	0.0	53.7	0.6	7.7	**67.3**	
	large-play-v2	0.0	**28.0**	0.2	0.0	0.0	15.8	0.0	3.5	**32.4**	
	large-diverse-v2	0.0	**34.3**	0.0	0.0	0.0	14.9	0.2	3.7	**36.9**	
	antmze-v2 average	16.7	41.4	27.3*	20.9*	11.4	**50.6***	19.8	25.6	**53.5**	
D4RL Kitchen	kitchen-complete-v0	**65.0**	4.0			11.8	43.8		1.5	50.2	
	kitchen-mixed-v0	51.5	40.0			24.2	51.0		1.1	**60.3**	
	kitchen-partial-v0	38.0	**66.0**			20.8	49.8		0.5	51.4	
	kitchen-v0 average	**51.5**	36.7			18.9	48.2		1.0	**54.0**	

- We propose studying if conditioning on goals can help provide compositionality in space just as the Bellman backup provides compositionality in time

Suite	Environment	BC	Filt. BC	TD3+BC	Onestep	CQL-r	CQL-p	DT	RvS-R	RvS-G	GCSL
D4RL Gym	halfcheetah-random-v2	2.3	2.0	11.0	6.9	**18.6**		2.2	3.9		
	hopper-random-v2	**4.8**	4.1	**8.5**	**7.8**	**9.3**		**7.5**	**7.7**		
	walker2d-random-v2	**1.7**	**1.7**	**1.6**	**6.1**	**2.5**		**2.0**	-0.2		
	random-v2 average	2.9	2.6	**7.0**	**6.9**	**10.1**		3.9	3.8		
	halfcheetah-medium-replay-v2	36.6	40.6	**44.6**	**42.4**	**47.3**		36.6	38.0		
	hopper-medium-replay-v2	18.1	75.9	60.9	71.0	**97.8**		82.7	73.5		
	walker2d-medium-replay-v2	26.0	62.5	**81.8**	71.6	**86.1**		66.6	60.6		
	medium-replay-v2 average	26.9	59.7	62.4	61.7	**77.1**		62.0	57.4		
	halfcheetah-medium-v2	42.6	42.5	48.3	**55.6**	49.1		42.6	41.6		
	hopper-medium-v2	52.9	56.9	59.3	**83.3**	64.6		67.6	60.2		
	walker2d-medium-v2	75.3	75.0	**83.7**	**85.6**	**82.9**		74.0	71.7		
	medium-v2 average	56.9	58.1	63.8	**74.8**	65.5		61.4	57.8		
	halfcheetah-medium-expert-v2	55.2	**92.9**	**90.7**	**93.5**	85.8		86.8	**92.2**		
	hopper-medium-expert-v2	52.5	**110.9**	98.0	102.1	102.0		**107.6**	101.7		
	walker2d-medium-expert-v2	**107.5**	**109.0**	**110.1**	**110.9**	**109.5**		**108.1**	**106.0**		
	medium-expert-v2 average	71.7	**104.3**	**99.6**	**102.2**	99.1		**100.8**	**100.0**		
	gym-v2 average	39.6	56.2	**58.2**	**61.4**	**63.0**		57.0	54.7		

Analysis of reward conditioning

- RvS-G can successfully reach many different goals entirely from randomly collected offline data



Analysis of reward conditioning

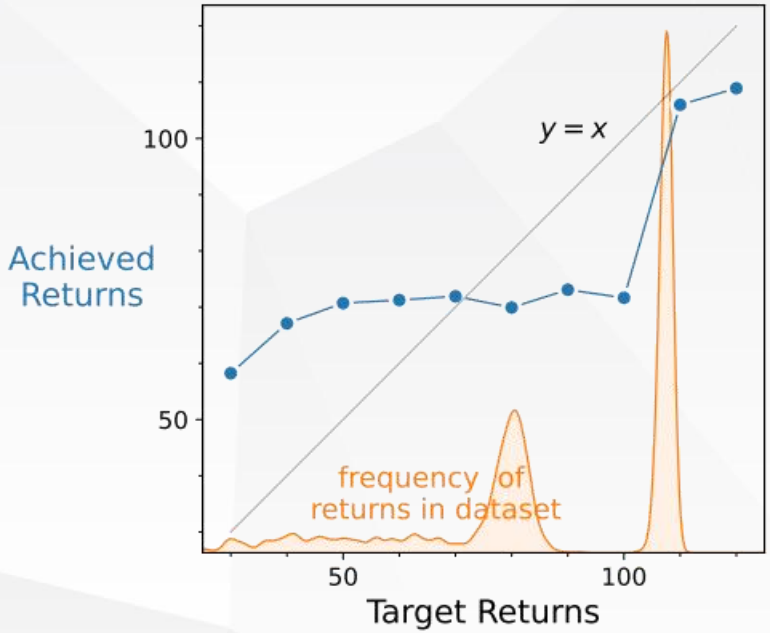


Figure 6: **A failure to interpolate.**

- The policy can't interpolate between the two modes, it appears that RvS is mimicking a subset of the demonstrations
- This analysis highlights the importance of the conditioning variable.

The End

thanks