Demonstration-Conditioned Reinforcement Learning for Few-Shot Imitation: Supplementary Material

1. Introduction

This document has a similar order to the main paper. We begin with the DCRL method (Section 2), discussing the equivalence between few-shot imitation agents and demonstration-conditioned policies, and the complexity of encoder layers with axial attention. Then we present our experimental settings (Section 3), including all hyperparameters for the proposed transformer-based policy network, and all training procedures. Finally, we give details of the Markov decision processes, task-by-task breakdowns of the returns and success rates, and t-SNE plots of the representations learned, for all experiments with the Meta-World manipulation benchmark (Section 4) and the navigation benchmark (Section 5).

2. Method

2.1. Equivalence of α and π

In Section 3.1 of the main paper, we stated that few-shot imitation agents are equivalent to demonstration-conditioned policies, under certain assumptions. Here, we explain this statement in more detail, using notation defined in Section 3 of the main paper.

Consider an agent $\alpha : \mathbf{D} \to \Pi$ and a demonstration-conditioned policy that assigns probability mass or density $\pi(a|h, \mathbf{d})$ to action $a \in \mathcal{A}$, given history $h \in \mathcal{H}$ and collection of demonstrations $\mathbf{d} \in \mathbf{D}$. As $\alpha(\mathbf{d})$ is a policy, it follows that $\alpha(\mathbf{d})(h)$ is a distribution over actions, for any history h. So, informally one might think of $\alpha(\mathbf{d})(h)$ and $\pi(\cdot|h, \mathbf{d})$ as being related by *currying* the argument h[1].

Formally, a probability distribution is a measure that assigns a number in $[0, 1]$, to any measurable set M, which represents the probability that the outcome is in set M. In our case, $\alpha(\mathbf{d})(h)(M)$ is the probability that the agent takes an action that lies in a measurable set $M \subseteq \mathcal{A}$. If the set of actions $\mathcal{A}$ is a finite set, then the distribution defines a probability mass function, so the equivalence is $\alpha(\mathbf{d})(h)(a) = \pi(a|h, \mathbf{d})$. However, in many real-world problems, it is natural to think of $\mathcal{A}$ as an infinite set. For instance, $\mathcal{A}$ might be the set of generalized forces that may be applied by a robot's actuators. Now, a distribution over an infinite set might have neither a probability density nor a probability mass function, one well-known example of such a distribution being the Cantor distribution. For those agents α whose distributions *are* given by a probability density function, the equivalence is that $\alpha(\mathbf{d})(h)(M) = \int_M \pi(a|h, \mathbf{d})\, da$ for any measurable set $M \subseteq \mathcal{A}$.

2.2. Complexity of the Encoder Layers with Axial Attention

Standard Encoder Layer. The complexity of the computations in a single head of an encoder layer, with no axial attention, may be decomposed as follows. We consider an input $X \in \mathbb{R}^{m \times H}$, weights $W^Q, W^K, W^V \in \mathbb{R}^{H \times H}$, intermediate variables T_1, T_2, and output Y.

Variable	Operations	Memory
$Q = XW^Q$	$O(mH^2)$	$m \times H$
$K = XW^K$	$O(mH^2)$	$m \times H$
$V = XW^V$	$O(mH^2)$	$m \times H$
$T_1 = QK^\top/\sqrt{H}$	$O(m^2H)$	$m \times m$
$T_2 = \text{softmax}(T_1)$	$O(m^2)$	$m \times m$
$Y = T_2V$	$O(m^2H)$	$m \times H$

[1]Currying means converting a function that takes multiple arguments, into a sequence of functions that each take a single argument, for example, $f(a, b, c)$ becomes $g(a)(b)(c)$

Thus, the total number of operations is $O(Hm(H+m))$. For backpropagation we must store X, W^Q, W^K, W^V and the other intermediate results, so the memory required for backpropagation is $O((H+m)^2)$, if we treat the batch size as fixed.

Axial Attention. Now let us find the complexity of applying a single head of an encoder layer with axial attention to a multi-dimensional input $\tilde{X}$, resulting in a multi-dimensional output $\tilde{Y}$, where $\tilde{X}, \tilde{Y} \in \mathbb{R}^{\ell_0 \times \ldots \ell_{d-1} \times H}$. For each dimension k where $0 \leq k < d$, we proceed as follows.

```
**for** each value of the indices $i_0, \ldots, i_{k-1}, i_{k+1}, \ldots, i_{d-1}$
    **let** $X \in \mathbb{R}^{\ell_k \times H}$ be the submatrix of $\tilde{X}$ with components $X_{i_k j} = \tilde{X}_{i_0 \ldots i_{d-1} j}$
    **let** $Y$ be the result of applying the above standard encoder layer calculation to $X$
    **set** the submatrix $[\tilde{Y}_{i_0 \ldots i_{d-1} j}]_{0 \leq i_k < \ell_k, 0 \leq j < H}$ equal to $Y$
```

Applying the result for the standard encoder layer given above, with $m = \ell_k$, multiplying by the number of values for the indices other than i_k, then summing over k gives the total complexity for an encoder layer with axial attention. The total number of operations is $O\big(H\big(\prod_{i=0}^{d-1} \ell_i\big)\big(Hd + \sum_{j=0}^{d-1} \ell_j\big)\big)$, and the memory for backpropagation is $O\big(\sum_{k=0}^{d-1} \big(\prod_{i:0 \leq i < d, i \neq k} \ell_i\big)(H + \ell_k)^2\big)$. If H is fixed, both complexities simplify to $O((\prod_{i=0}^{d-1} \ell_i) \sum_{j=0}^{d-1} \ell_j)$, which compares favourably to the $O(\prod_{i=0}^{d-1} \ell_i^2)$ complexity without axial attention.

2.3. Detailed Network Architecture

Figure 2 of the main paper showed a simplified view of our proposed demonstration-conditioned transformer-based policy network. Figure 1 shows that architecture in more detail.

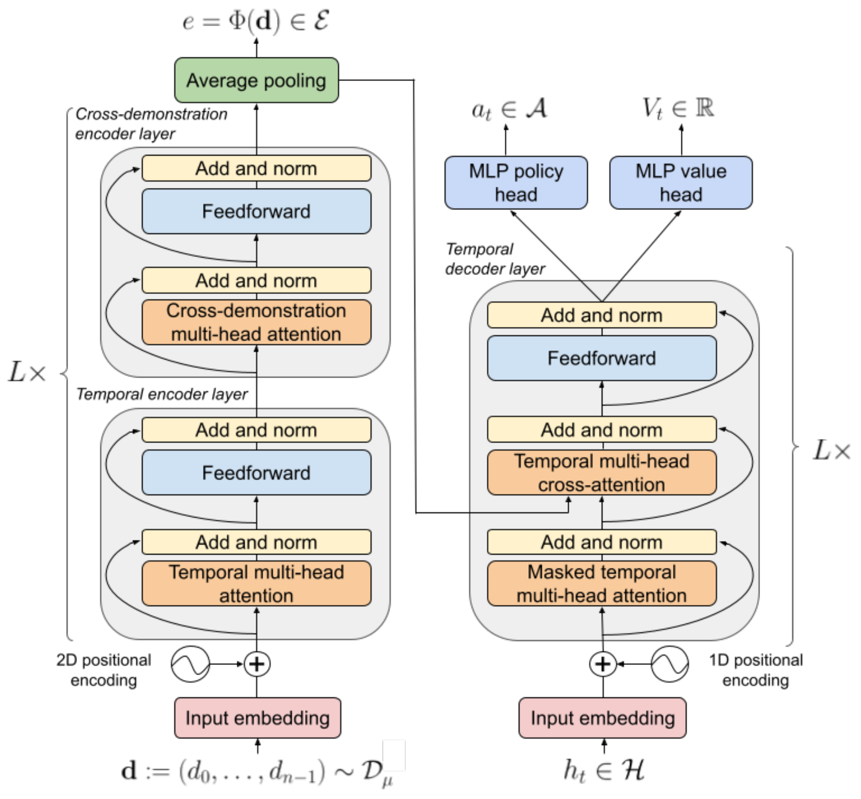


Figure 1. Architecture of proposed transformer-based policy with axial attention.

3. Experimental Settings

3.1. Hyperparameters

Table 1. Hyperparameters of the proposed transformer-based policy network.

Hyperparameter	Value
Hidden dimension, H	64
Number of heads	8
Feedforward dimension	512
Activation function	ReLU
Number of cross-demonstration encoder layers, L	2
Number of temporal encoder layers, L	2
Number of decoder layers, L	2
Number of policy layers	3
Number of hidden units in policy layers	64, 64, dim$(\mathcal{A})$
Number of critic layers	3
Number of hidden units in critic layers	64, 64, 1

Table 1 presents the hyperparameters of the proposed transformer-based architecture, used in our experiments with DCRL and the DCBC baselines. Compared with transformer-based language models (Vaswani et al., 2017; Brown et al., 2020), the proposed model is small, especially in its hidden dimension H and the number of layers L. There is room for improvement, as we did not tune these hyperparameters.

Table 2. Parameters of the PPO algorithm used to train DCRL. The clipping parameter ϵ is that which appears in equation (7) of Schulman et al. (2017). GAE is an abbreviation for generalized advantage estimation.

Hyperparameter	Value
Clipping, ϵ	0.2
Discount factor, γ	0.99
GAE parameter, λ	0.95
Epochs	6
Learning rate schedule	linear annealing
Value clipping	no
Entropy coefficient	1e-3
Value coefficient	0.5
Optimizer	AdamW
L2 regularization coefficient	1e-4
Learning rate	3e-4
Update time-step	1024
Batch size	256
Gradient-norm clipping	0.5

Tables 2, 3, 4 and 5 present the hyperparameters used when training DCRL, when training the behaviour-cloning baseline DCBC+MT, when meta-training and finetuning with REPTILE, and when training the task-specific expert policies from which we collected demonstrations, respectively. In all cases, weights were initialized by the default PyTorch method, with Kaiming initialization for linear layers (He et al., 2015) and Xavier initialization for transformer layers (Glorot & Bengio, 2010). DCRL requires about 250 million frames in total to train for each benchmark, using 45–50 training environments. This represents 5 million frames per environment which is about 4×10^4 episodes per training task. We did not perform hyperparameter tuning.

Table 3. Hyperparameters used to train the behaviour-cloning baseline DCBC+MT.

Hyperparameter	Value
Learning rate	1e-4
Batch size	256
Optimizer	AdamW
L2 regularization coefficient	1e-2
Early stopping	yes
Gradient-norm clipping	no

Table 4. Hyperparameters used when meta-training and finetuning with REPTILE.

Hyperparameter	Value
Learning rate	1e-4
Batch size	256
Meta-learning rate	1e-3
Inner updates	250
Outer updates	1000
Optimizer	AdamW
L2 regularization coefficient	1e-2
Early stopping	yes
Gradient-norm clipping	no

Table 5. Hyperparameters used when training task-specific policies with PPO. These policies are used to provide demonstrations.

Hyperparameter	Value
Clipping	0.2
Discount factor, γ	0.99
GAE parameter, λ	0.95
Update time-step	4096
Batch size	1024
Epochs	6
Learning rate	3e-4
Learning-rate schedule	linear annealing
Gradient norm clipping	0.5
Value clipping	no
Entropy coefficient	1e-3
Value coefficient	0.5
Number of linear layers	3
Hidden dimension	64
Activation function	$\tanh$
Optimizer	Adam

3.2. Experimental Statistics

We provide the standard deviations of all results that we report. Here, we explain how those standard deviations are computed, and which source of randomness they characterize.

To evaluate the performance of a model on a single task, we perform 300 episodes with that model in the task's environment. In each of these episodes, the collection of demonstrations and the initial state are sampled from their respective distributions. When reporting results for a single task, we give the mean and standard deviation of the return or success over these

300 episodes. When reporting results for multiple tasks, we give the mean of the single-task means, and the mean of the single-task standard deviations.

Thus, the reported standard deviations account for variability due to the initial state and the collection of demonstrations. However, we only use *one* policy when evaluating a model on a task, so the reported standard deviations do not capture variability due to the random nature of model initialisation and training. It is well known that this is a major source of variability in the performance of deep reinforcement learning models, and we consider investigating this source of variability in future work.

3.3. Finetuning Procedure

Sections 4.1.2 and 4.2.2 of the main paper report results of DCRL, DCBC+REPTILE and DCBC+MT, after finetuning. We now explain this finetuning procedure in detail.

Training Batch Generation. A single training sample is composed of a collection of demonstrations $\mathbf{d} \in \mathbf{D}$, and a history $h \in \mathcal{H}$, which includes demonstrator actions. When finetuning with $k \in \mathbb{Z}_{>0}$ demonstrations, we sample k distinct demonstrations $d_i := (o_0, o_1, \ldots, o_{T-1})$ for $i = 0, \ldots, k-1$, at random from the available $N = 5000$ demonstrations of the task at hand. Each observation consists of both a state and an action, so that $o_t \in \mathcal{S} \times \mathcal{A}$, and the length of each demonstration $T \in \mathbb{Z}_{>0}$ is random. From those demonstrations, we build collections of demonstrations, as follows. We compose a collection $\mathbf{d} = (d_0, \ldots, d_{n-1})$ by sampling uniformly (without replacement) a random number $n \sim \text{Uniform}(\{1, \ldots, k\})$ of demonstrations. With this protocol, one can build $\sum_{i=1}^{k} \frac{k!}{(k-i)!}$ different collections of demonstrations. We create a training sample from a collection of demonstrations $\mathbf{d}$ by uniformly sampling one of the k demonstrations to be used as the history h. Thus, in total, one can build $k \times \sum_{i=1}^{k} \frac{k!}{(k-i)!}$ distinct training samples, which is 1625 for $k = 5$.

Optimization. Finetuning is performed by optimising a standard behaviour-cloning loss with AdamW (cross-entropy loss if $\mathcal{A}$ is a finite set, and squared error otherwise). We use the same batch size for finetuning as we use for training, if the number of demonstrations per collection n is big enough to allow this.

Stopping Criterion. We stop the finetuning process when the training loss on the finetuning sample first goes below a threshold (0.05 for squared error and 0.2 for cross-entropy). Another solution would be to use early stopping based on a validation loss. However, we do not use much data when finetuning for a single task; thus, any validation set would be so small that the validation loss would not typically be a good estimate of the true out-of-sample loss.

4. Manipulation Benchmark

4.1. Settings

4.1.1. Task Splits

We randomly split the 50 Meta-World tasks into 10 folds. The folds used were as follows.

Fold 1. reach-v1, push-wall-v1, pick-out-of-hole-v1, coffee-pull-v1, plate-slide-side-v1.
Fold 2. reach-wall-v1, disassemble-v1, sweep-into-v1, handle-pull-side-v1.
Fold 3. pick-place-v1, door-close-v1, faucet-open-v1, sweep-v1, plate-slide-v1.
Fold 4. pick-place-wall-v1, faucet-close-v1, soccer-v1, handle-pull-v1, window-open-v1.
Fold 5. push-v1, assembly-v1, handle-press-v1, peg-unplug-side-v1, basketball-v1.
Fold 6. coffee-push-v1, box-close-v1, button-press-topdown-v1, drawer-open-v1, door-lock-v1.
Fold 7. coffee-button-v1, plate-slide-back-side-v1, window-close-v1, door-open-v1, 'button-press-v1.
Fold 8. hand-insert-v1, lever-pull-v1, drawer-close-v1, handle-press-side-v1, button-press-wall-v1.
Fold 9. hammer-v1, dial-turn-v1, plate-slide-back-v1.
Fold 10. shelf-place-v1, push-back-v1, peg-insert-side-v1, button-press-topdown-wall-v1.

4.1.2. States and Actions

The state is of the form $s = (s_h, s_o, s_{o'}, s_g, s_{\text{gripper}}) \in \mathbb{R}^{13}$, where $s_h, s_o, s_{o'}, s_g \in \mathbb{R}^3$ are the Cartesian coordinates of the end-effector, the first object, the second object (if there is one, else zero) and the goal respectively, and $s_{\text{gripper}} \in \mathbb{R}$ is the state of the gripper. The actions are of the form $a = (a_h, a_{\text{gripper}}) \in [-1,1]^4$ where $a_h \in [-1,1]^3$ is the desired Cartesian coordinates of the end-effector at the next time, and $a_{\text{gripper}} \in [-1,1]$ is the desired gripper state.

4.1.3. Reward Functions

Original Reward Function. For every task μ in the Meta-World benchmark, the reward function specified in the Yu et al. (2019b) has one of the following three forms:

$$R_{\text{MW},\mu}(s,\cdot) = \begin{cases} R_{\text{push}}(s,\cdot), \\ R_{\text{push}}(s,\cdot) + R_{\text{reach}}(s,\cdot), \\ R_{\text{reach}}(s,\cdot) + R_{\text{grasp}}(s,\cdot) + R_{\text{place}}(s,\cdot), \end{cases}$$

where we write $(s,\cdot)$ as the reward function is independent of the action. The four terms are defined as

$$\begin{aligned} R_{\text{reach}}(s,\cdot) &:= -\left\|s_h - s_o\right\|_2, \\ R_{\text{grasp}}(s,\cdot) &:= c_1\, \mathbf{1}_{\|s_h - s_o\|_2 < \epsilon}\, \min\{s_{o,z}, z_{\text{target}}\}, \\ R_{\text{push}}(s,\cdot) &:= c_2\, \mathbf{1}_{\|s_h - s_o\|_2 < \epsilon}\, \exp(-\left\|s_o - s_g\right\|_2^2 / c_3), \\ R_{\text{place}}(s,\cdot) &:= c_2\, \mathbf{1}_{\|s_{o,z} - z_{\text{target}}\| < \epsilon}\, \exp(-\left\|s_o - s_g\right\|_2^2 / c_3), \end{aligned}$$

where $c_1 = 100, c_2 = 1000, c_3 = 0.01, \epsilon = 0.05$, and $\mathbf{1}_.$ is the indicator function, noting that the minuses in the exponents of R_{push} and R_{place} are missing in the Meta-World paper but not in the corresponding source code, and that the source code does not always follow this specification exactly. A task is considered as successfully completed if $\left\|s_o - s_g\right\|_2 < \epsilon_{\text{success},\mu}$, where $\epsilon_{\text{success},\mu} \in \{0.02, 0.04, 0.05, 0.07, 0.08\}$ is a task-dependent threshold, although again the available and official source code does not follow this precisely.

Unfortunately, the reward function $R_{\text{MW},\mu}$ can make it preferable for agents to stay in regions of high reward, than to successfully complete a task. For instance, say the reward is of the form $R_{\text{push}} + R_{\text{reach}}$, and the success threshold is $\epsilon_{\text{success},\mu} = 0.02$, then for $\left\|s_o - s_g\right\|_2 = 0.03$ there is a reward of up to $c_2 \exp(-0.03^2/0.01) > 900$, and the episode does not terminate. The return is therefore maximized if the agent's keeps its state such that $\left\|s_o - s_g\right\|_2$ is as low as possible but never less than 0.02. A concrete example of this problem in the actual source code[2] is for the reach-v1 task, where the reward is

$$R_{\text{MW},\mu}(s_t,\cdot) = \max\{0, c_1\,(d_{\text{reach}}(s_0) - d_{\text{reach}}(s_t) + \exp(-d_{\text{reach}}(s_t)^2/c_2) + \exp(-d_{\text{reach}}(s_t)^2/c_3))\}, \tag{1}$$

where $d_{\text{reach}}(s) := \left\|s_h - s_g\right\|_2$ and the success criterion is $d_{\text{reach}}(s_t) \leq 0.05$.

Modified Reward Function. To overcome this undesirable behaviour, we use a modified reward function in this paper, which acts like the time derivative of the original reward, and which is given by

$$R_\mu(s,a) = \begin{cases} 0 & \text{if } s \text{ is the last state of the episode} \\ R_{\text{MW},\mu}(f_{\text{MuJoCo}}(s,a),\cdot) - R_{\text{MW},\mu}(s,\cdot) & \text{otherwise,} \end{cases}$$

where $f_{\text{MuJoCo}}(s,a)$ is the next state as determined by MuJoCo.

To demonstrate that the modified reward function overcomes the undesirable behaviour, we compare two episodes, with identical histories up to time $T-2$. At time $T-2$, the agent has the choice of succeeding by moving into a terminal state s_{terminal}, or moving to a nearby non-terminal state $s_{\text{non-terminal}}$ for $k > 0$ steps before going to s_{terminal}. Let $\bar{J}_T$ and $\bar{J}_{T+k}$ be the returns of these episodes, so that

$$\bar{J}_T = \sum_{t=0}^{T-1} \gamma^t R_\mu(s_t, a_t) = \gamma^{T-1} R_{\text{MW},\mu}(s_{\text{terminal}},\cdot) - R_{\text{MW},\mu}(s_0,\cdot) + (1-\gamma)\sum_{t=1}^{T-2} \gamma^t R_{\text{MW},\mu}(s_t,\cdot).$$

Let us assume that $\gamma \in (0,1)$, as in our experiments, and that $R_{\text{MW},\mu}(s_{\text{non-terminal}},\cdot) =: \bar{R} > 0$ and $R_{\text{MW},\mu}(s_{\text{terminal}},\cdot) =: (1+\delta)\bar{R}$ with $\delta > 0$, which holds for the reward (1) for states that satisfy or nearly satisfy the success criteria. Immediate

[2]Referring to the file https://github.com/rlworkgroup/metaworld/blob/master/metaworld/envs/mujoco/sawyer_xyz/v1/sawyer_reach_push_pick_place.py, last retrieved on 8th February 2021.

termination is strictly preferable if $\bar{J}_T > \bar{J}_{T+k}$, which holds if

$$\gamma^{T-1}\bar{R}(1+\delta) > \gamma^{T+k-1}\bar{R}(1+\delta) + (1-\gamma)\sum_{t=T-1}^{T+k-2}\gamma^t\bar{R} \qquad \Leftrightarrow \qquad 1+\delta > \gamma^k(1+\delta) + (1-\gamma)\sum_{t=0}^{k-1}\gamma^t,$$

which is true as $\delta > 0$ and $\gamma < 1$. Thus, the modified reward function makes it preferable for an agent to successfully complete tasks, than to delay their completion.

4.2. Few-Shot Imitation

4.2.1. Discounted Returns and Success Rates for Each Task

Figures 2 and 3 give the discounted return and success rate on each Meta-World task for the few-shot imitation experiment (Section 4.1.1 of the main paper).

4.2.2. Representation Learning

We now study how the number of training tasks relates to the final few-shot imitation performance. Figures 4 and 5 compare the discounted return and success rate of DCRL when trained with 10, 20 and 40 training tasks. To pick the training tasks for a model that is applied to the test tasks of a given fold, we randomly sampled tasks from the other folds without replacement (the folds were specified in Section 4.1.1).

As expected, both performance metrics improve as the number of training tasks increases, on average over tasks, for each number of demonstrations. It will be interesting to train DCRL on larger collections of tasks in future.

4.2.3. t-SNE of Demonstrations and Their Embeddings

In Section 3.2 of the main paper, we suggested that generalization to new tasks might be interpreted as interpolation in an embedding space, where demonstrations are close if and only if the tasks have similar optimal policies. To begin exploring this notion, Figure 6 presents visualizations of demonstrations and embedded demonstrations using t-SNE (van der Maaten & Hinton, 2008), a popular nonlinear dimensionality-reduction tool. For each task, we extracted 500 demonstrations using task-specific PPO policies, and made 500 corresponding collections of demonstrations, each consisting of a single demonstration. Each demonstration is of size $T \times 13$, where the length T varies from one demonstration to another, and 13 is the dimension of the Meta-World state observations. As t-SNE expects fixed-size vectors as input, we used the time-average of each demonstration as the input for the plot t-SNE$(\mathbf{d})$, which is thus a projection from $\mathbb{R}^{13}$ to $\mathbb{R}^2$. As we are interested in representation learning, we compare two embedding functions: Φ_{random} uses weights set to their random initial values before training; and Φ_{learned} is an embedding trained on 9 out of the 10 folds. For our transformer-based policy architecture, these embeddings are of size $T \times 64$, so we also took their time averages; thus, the plots of t-SNE$(\Phi_{\text{random or learned}}(\mathbf{d}))$ are projections from $\mathbb{R}^{64}$ to $\mathbb{R}^2$.

In each plot, the points are colour-coded by task identity, and different tasks form different clusters even though the task identity was not provided to t-SNE. In the plot t-SNE$(\mathbf{d})$, the clusters for the tasks sweep-into-v1, push-wall-v1, push-v1, coffee-push-v1 and soccer-v1 overlap. However, after embedding with Φ_{random}, there is no overlap between any tasks. Looking at t-SNE$(\Phi_{\text{learned}}(\mathbf{d}))$, we see that learning has brought together clusters for semantically similar tasks, including: button-press-topdown-v1 with button-press-topdown-wall-v1; reach-v1 with reach-wall-v1; and pick-place-v1 with pick-place-wall-v1. The pairs of tasks that have been brought together should indeed have similar optimal policies, supporting our notion of how generalization to new tasks might arise. We observe that the clusters for the learned embedding tend to be more elongated and are sometimes more fragmented than for the randomly-initialized embedding, but we do not have an explanation for this observation.

As these results show that a single demonstration should be enough to reliably distinguish between Meta-World tasks, we should not expect cross-demonstration attention to improve performance much for the Meta-World benchmark. As we wished to explore cross-demonstration attention, this motivated us to introduce the navigation benchmark.

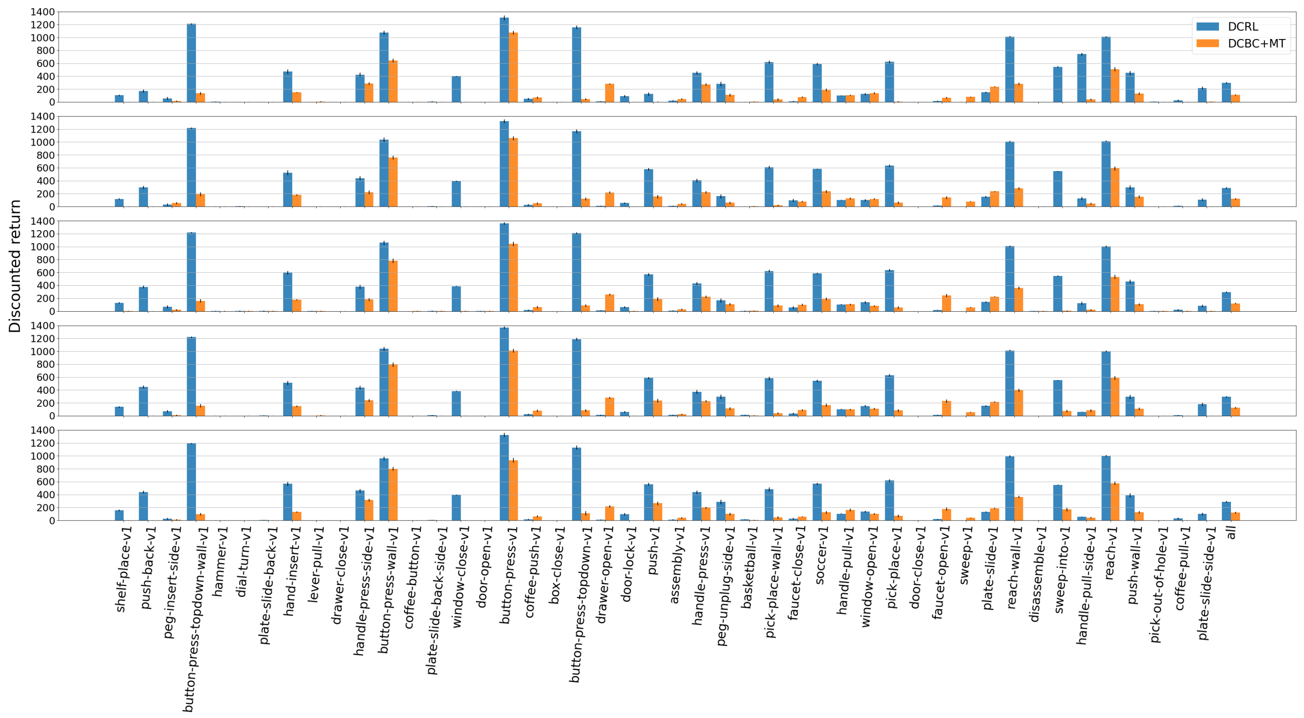


Figure 2. Discounted return of DCRL and DCBC+MT on each Meta-World task in the few-shot experiment for one demonstration (top row) to five demonstrations (bottom row).

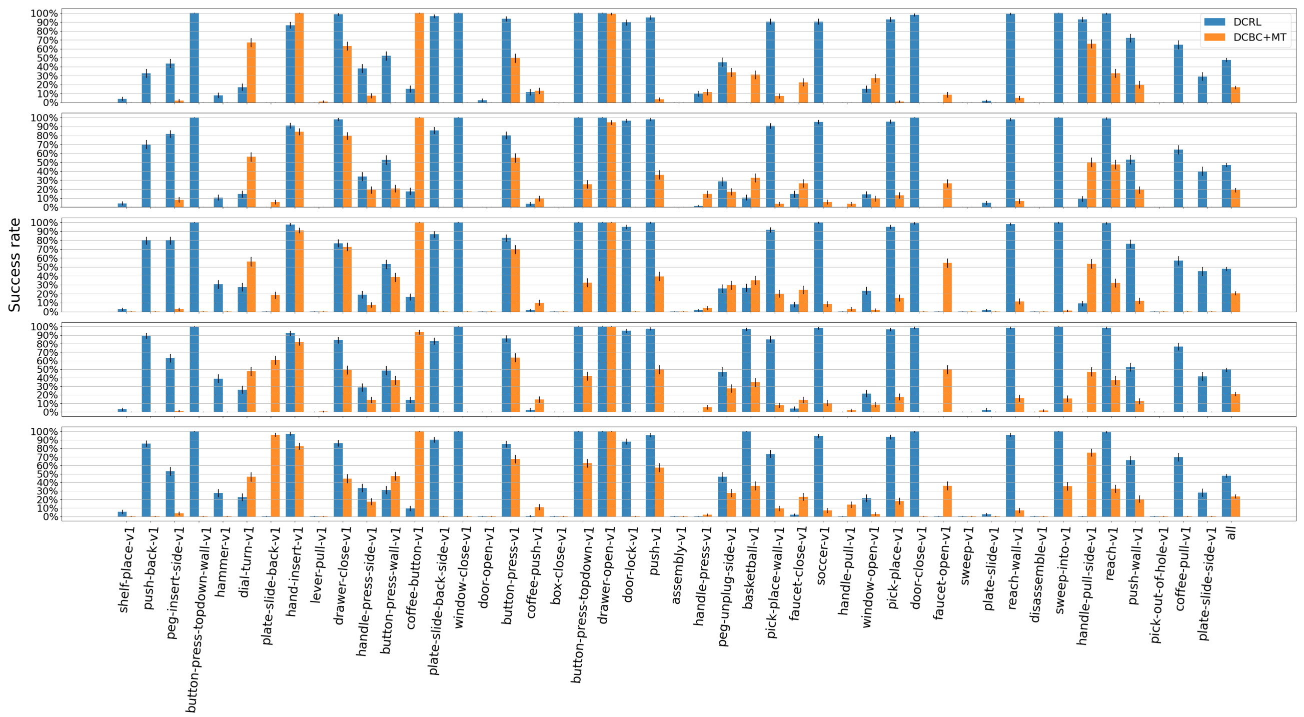


Figure 3. Success rate of DCRL and DCBC+MT on each Meta-World task in the few-shot experiment for one demonstration (top row) to five demonstrations (bottom row).

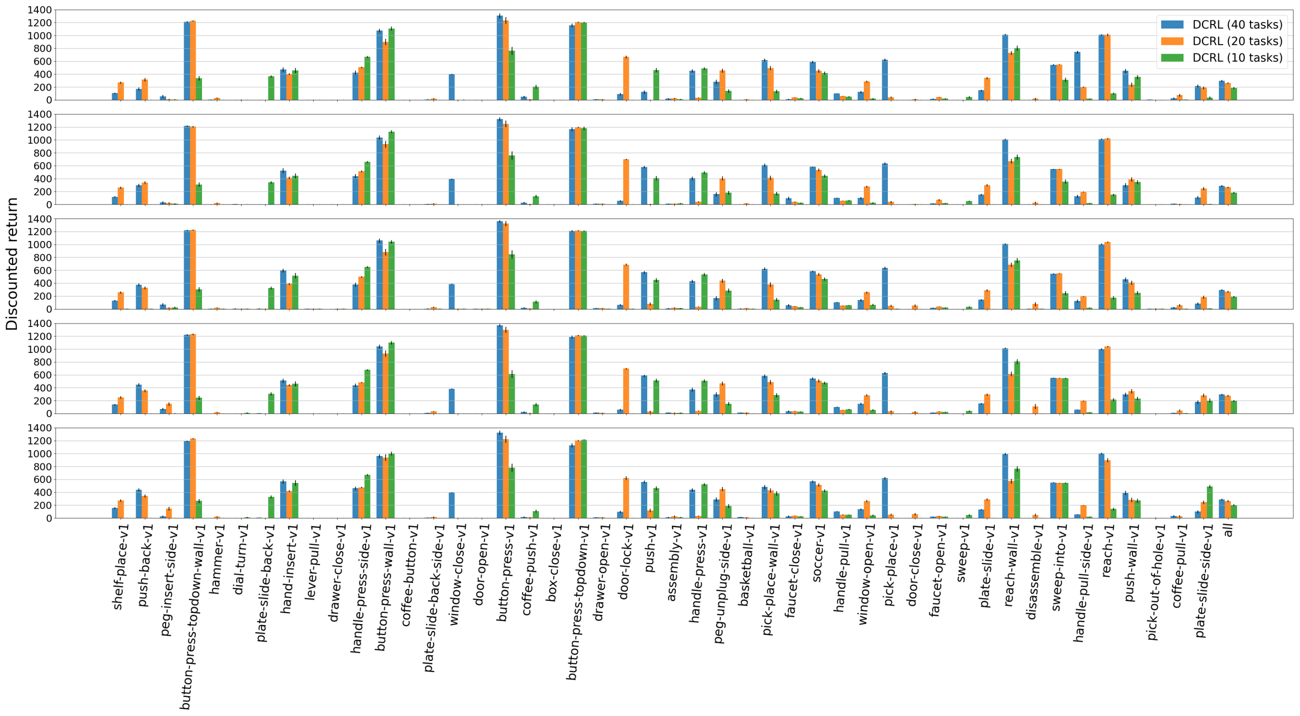


Figure 4. Discounted return of DCRL using 40 (blue), 20 (orange) and 10 (green) training tasks, on each Meta-World task in the few-shot experiment for one demonstration (top row) to five demonstrations (bottom row).

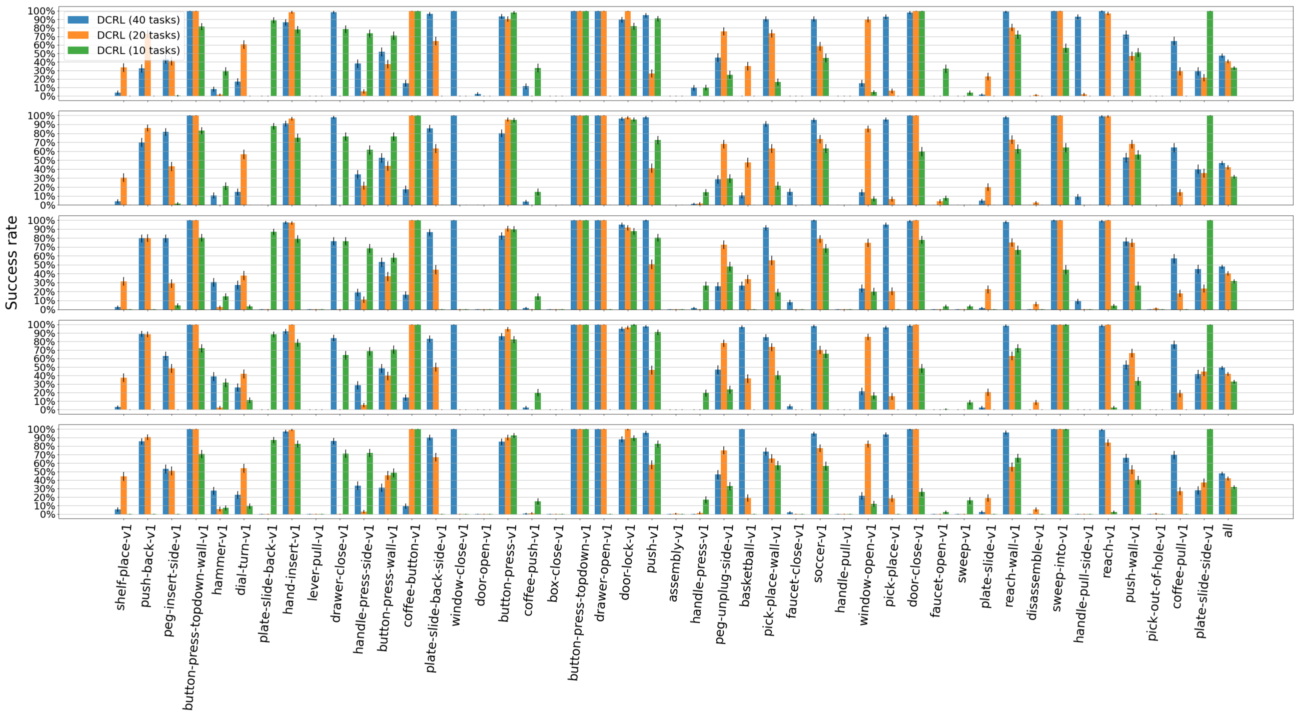


Figure 5. Success rate of DCRL using 40 (blue), 20 (orange) and 10 (green) training tasks, on each Meta-World task in the few-shot experiment for one demonstration (top row) to five demonstrations (bottom row).

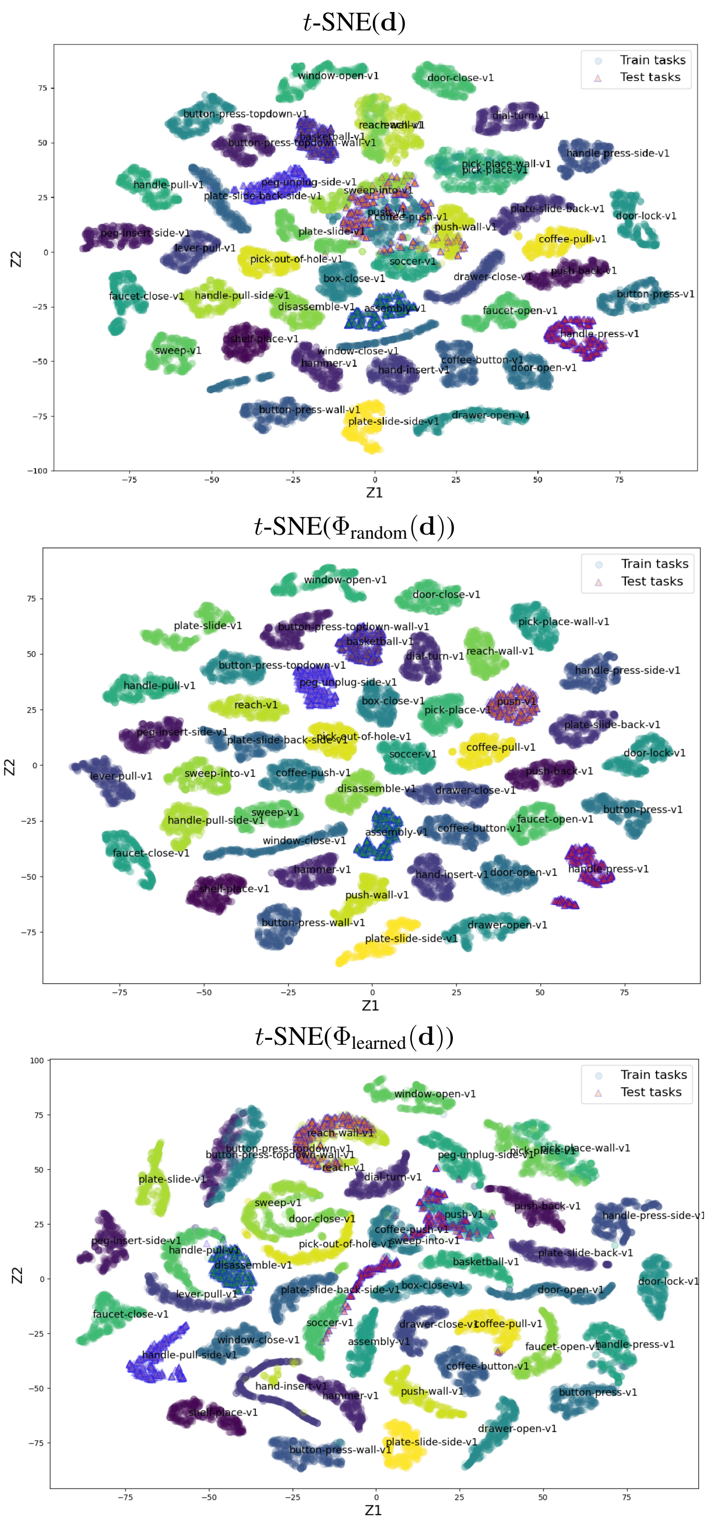


Figure 6. t-SNE plots of time-averaged demonstrations (top), and of a random embedding $\Phi_{\text{random}}(\mathbf{d})$ (middle) and a learned embedding $\Phi_{\text{learned}}(\mathbf{d})$ (bottom) of those demonstrations. Each point corresponds to a single demonstration, and points are colour-coded by task identity. Triangles represent test tasks, while circles represents training tasks. ($Z1$ and $Z2$ are simply names for the two dimensions of t-SNE's output.)

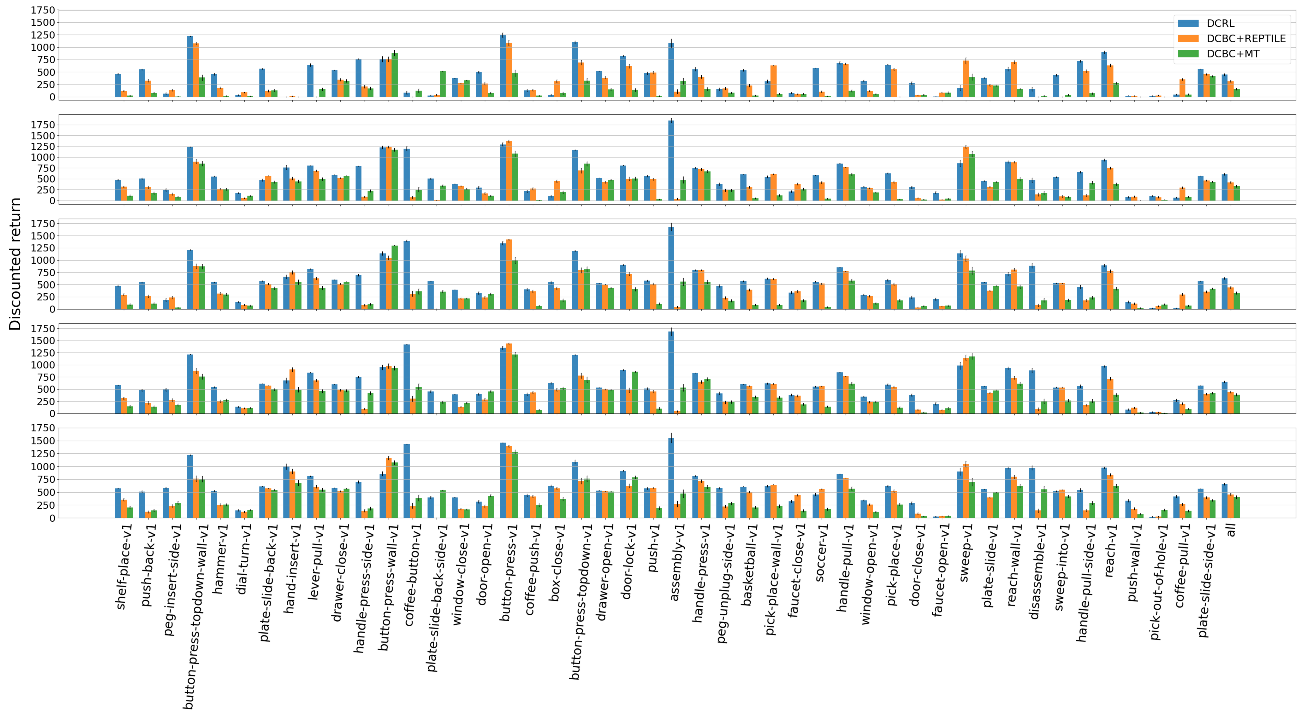


Figure 7. Discounted return of DCRL, DCBC+MT and DCBC+REPTILE on each Meta-World task, in the finetuning experiment for one demonstration (top row) to five demonstrations (bottom row).

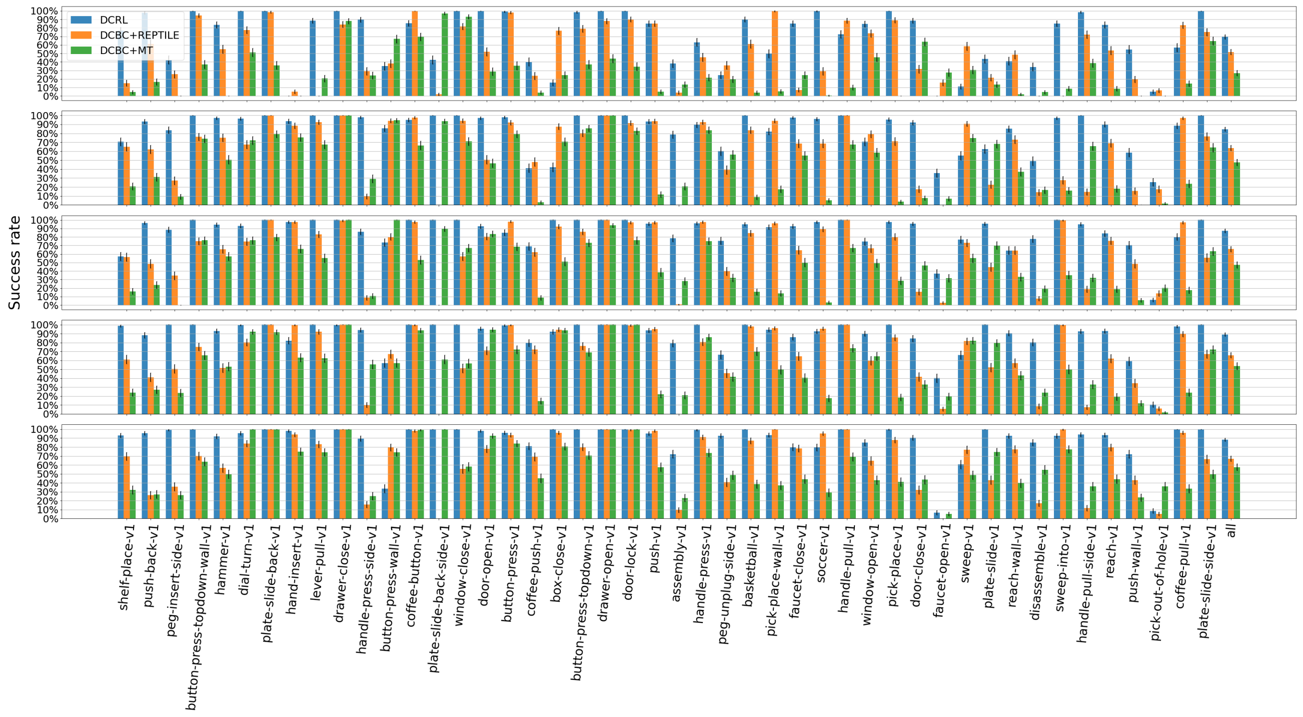


Figure 8. Success rate of DCRL (blue), DCBC+MT (green) and DCBC+REPTILE (orange) on each Meta-World task, in the finetuning experiment for one demonstration (top row) to five demonstrations (bottom row).

4.3. Few-Shot Imitation With Finetuning

4.3.1. Discounted Returns and Success Rates for Each Task

Figures 7 and 8 compare the discounted return and success rate on each Meta-World task, for the finetuning experiment (Section 4.1.2 of the main paper). With five demonstrations, DCRL attains a 90% success rate averaged over tasks. Interestingly, in this context DCRL successfully completes most of the tasks but only achieves a 7% success rate on faucet-close-v1 and a 9% success rate on pick-out-of-hole-v1.

4.4. Domain Shift

4.4.1. Meta-World With Sawyer and LIMS2-AMBIDEX

In Section 4.1.3 of the main paper, we compared the performance of DCRL when given demonstrations from a Rethink Robotics Sawyer robot, with its performance given demonstrations from a LIMS2-AMBIDEX robot. Both robots are 7-DOF serial manipulators equipped with a gripper as end-effector. While different robots provide demonstrations, in both cases, the robot performing the task was a Sawyer. Figure 9 shows four Meta-World tasks performed by a task-specific PPO policy on both the Sawyer and on LIMS2-AMBIDEX.

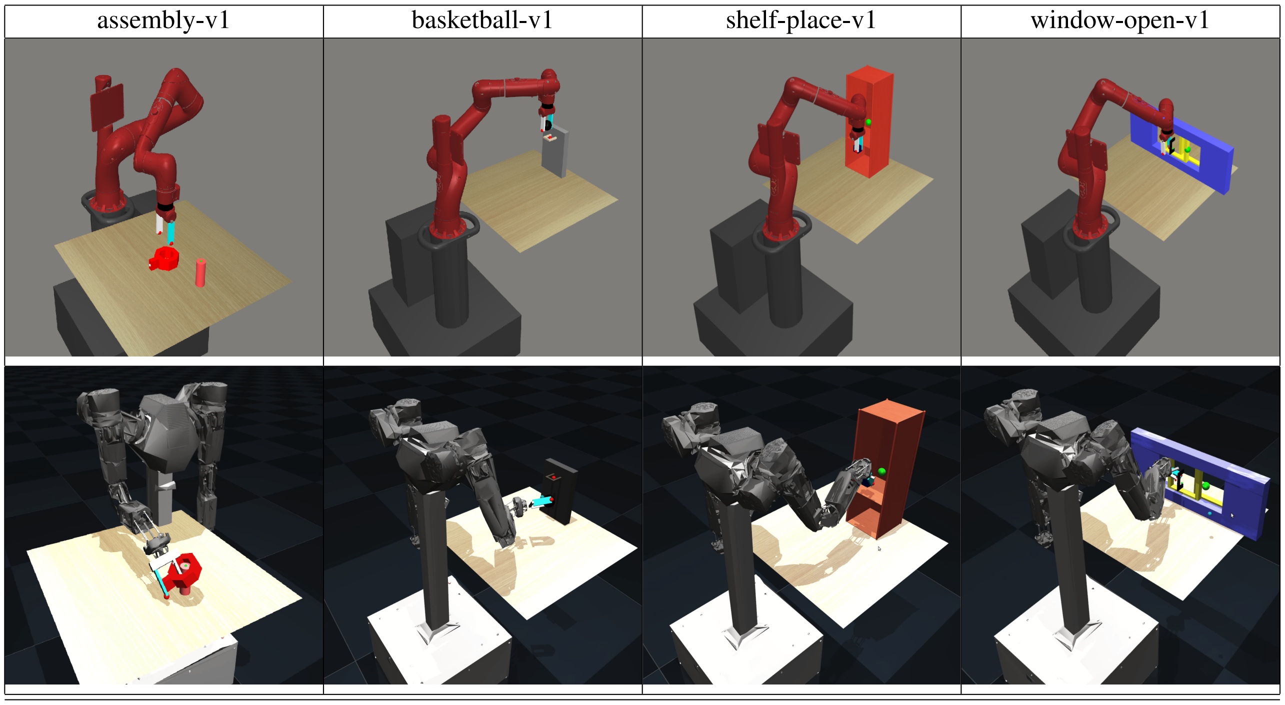


Figure 9. Four Meta-World tasks performed by a Sawyer robot (top row) and a LIMS2-AMBIDEX robot (bottom row).

4.4.2. Discounted Returns and Success Rates for Each Task

Figures 10 and 11 show the discounted return and success rates for the demonstrator domain-shift experiment discussed in the Section 4.1.3 of the main paper. The performance of DCRL (Sawyer demonstrations) is similar to that of DCRL (AMBIDEX demonstrations) for most tasks, with neither method having a systematic advantage over the other, and the average returns and success rates for the two methods are very similar. These results support our claim that DCRL is able to learn useful few-shot imitation agents when there is a domain shift between the agent and the demonstrator.

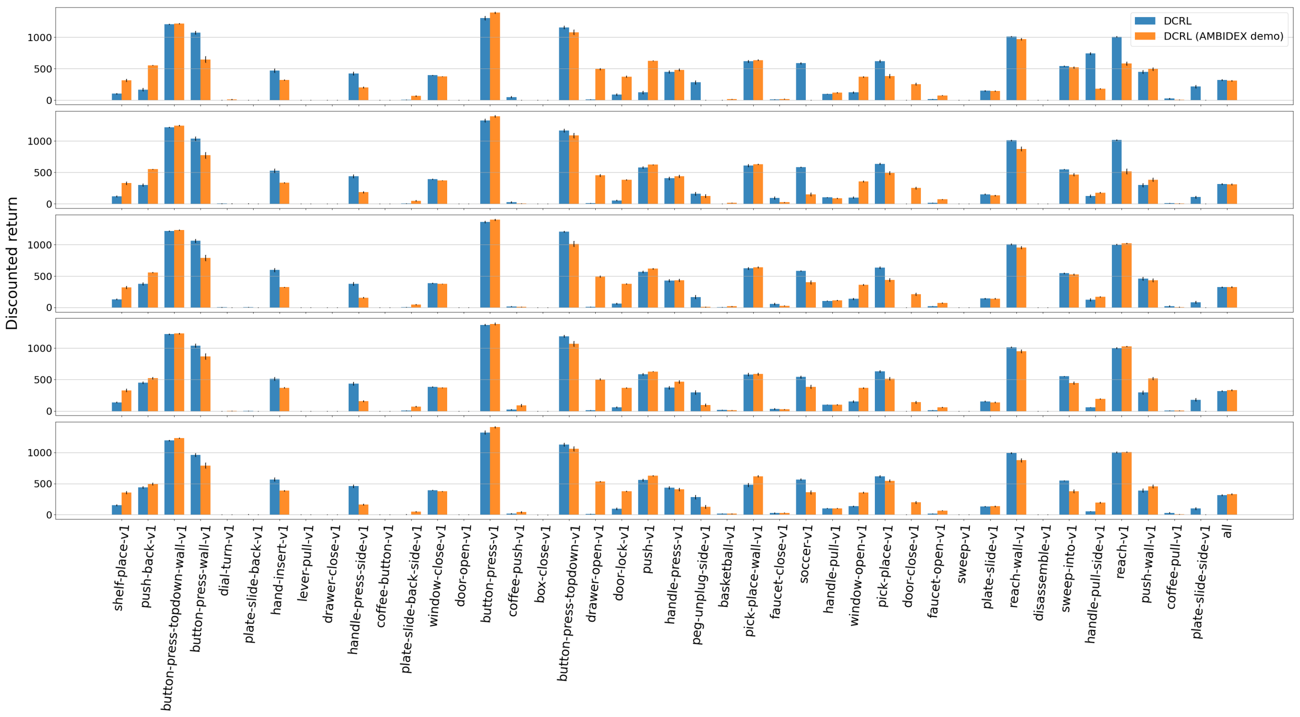


Figure 10. Discounted return of DCRL using demonstrations from Sawyer and AMBIDEX on each Meta-World task, in the few-shot experiment, for one demonstration (top row) to five demonstrations (bottom row).

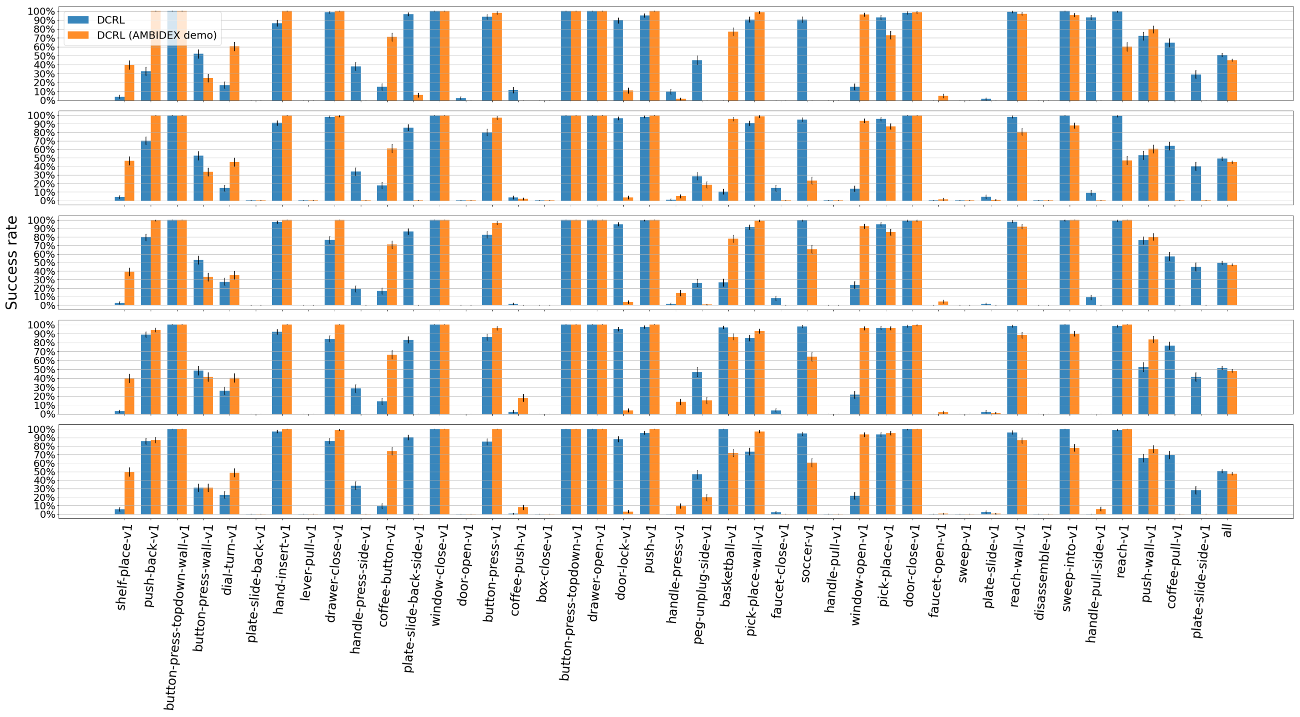


Figure 11. Success rate of DCRL using demonstrations from Sawyer and AMBIDEX on each Meta-World task in the few-shot experiment for one demonstration (top row) to five demonstrations (bottom row).

4.5. Improving on Suboptimal Demonstrations

4.5.1. Returns of Expert Against DCRL

Figure 4 in Section 4.1.4 of the main paper compares the success rates of perturbed demonstrators and DCRL. Figure 12 gives the analogous result for the discounted return. For $\sigma = 1$, experts achieve an averaged discounted return of about 490 while Figure 7 shows that a finetuned DCRL using non-perturbed demonstrations achieves a higher average return.

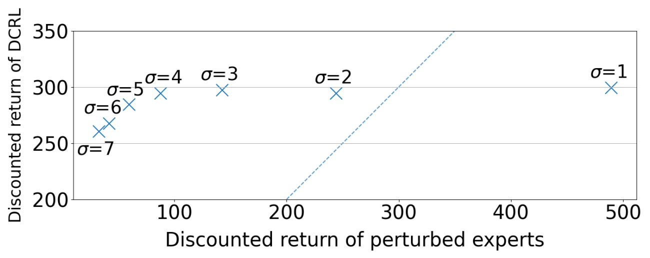


Figure 12. Discounted return of perturbed experts against the return of DCRL, using three demonstrations from perturbed demonstrators as input. Return is averaged over all Meta-World tasks.

4.6. Robustness to Random Downsampling

In this section, we assess the robustness of DCRL to perturbations of its input demonstrations. We consider a *demonstration dropout* perturbation, which consists in randomly dropping (that is, removing) selected time steps from individual demonstrations. This experiment uses the policies resulting from training without dropout — better results would be expected if the training demonstrations were perturbed. As expected, Figures 13 and 14 show that the performance of DCRL decreases as the dropout increases. However, most tasks are hardly affected even when 66% of steps are dropped, and the average success rate, which was 48% when no steps are dropped, remains above 40% as the number of input demonstrations ranges from one to five.

4.7. No-Demonstration Experiment

As the performance of DCRL on Meta-World hardly improves when given more than one input demonstration (Figure 2 and 3), it seems natural to ask how DCRL performs given *zero* input demonstrations. In view of previously reported success rates on Meta-World[3], it would also be natural to expect that such a no-demonstration policy would have a success rate of under 40%. It is therefore remarkable to see that such a no-demonstration policy achieves an average success rate of 45%, as shown in Figures 15 and 17. Ten versions of the no-demonstration policy were trained and tested on the same ten folds as the DCRL algorithm used in the other experiments. Therefore this success rate is directly comparable with the 48% success rate achieved by DCRL with one input demonstration.

To understand how the no-demonstration policy achieves such results, we looked at videos of it performing tasks. We see that it tends to execute a common sequence of steps for all tasks: move the end-effector to the first object; grasp the object; and finally, go to the goal. In tasks where there is no need to go to the object, following this sequence results in slower task completion, and this can be seen in the consistently lower returns for the no-demonstration policy on the tasks hand-insert-v1, reach-v1 and reach-wall-v1. The success of the no-demonstration policy shows that the tasks of the Meta-World benchmark are less diverse than they appear at first glance, and this motivates us to consider a more diverse benchmark in the next section.

4.8. No-History Experiment

We chose to use the agent's full history as input to DCRL and the DCBC-based methods. The Meta-World tasks are indeed POMDPs — for instance, the robot arm has momentum, but we only observe position information. However, it is interesting

[3]Table 1 of Yu et al. (2019b) gives success rates of 23.9%, 20% and 30% for MAML, RL^2 and PEARL respectively, for the ML45 setting at meta-test time.

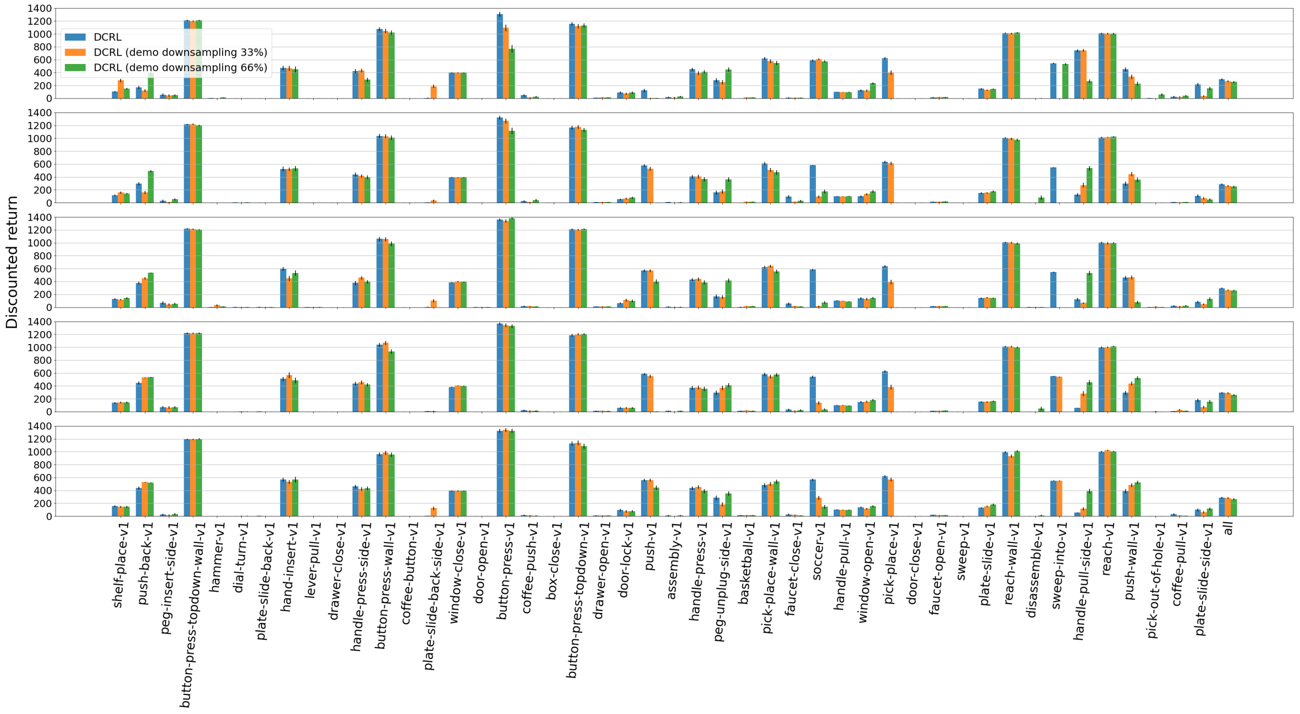


Figure 13. Discounted return of DCRL using a demonstration dropout of 0%, 33% and 66%, on each Meta-World task in the few-shot experiment, for inputs consisting of one demonstration (top row) to five demonstrations (bottom row).

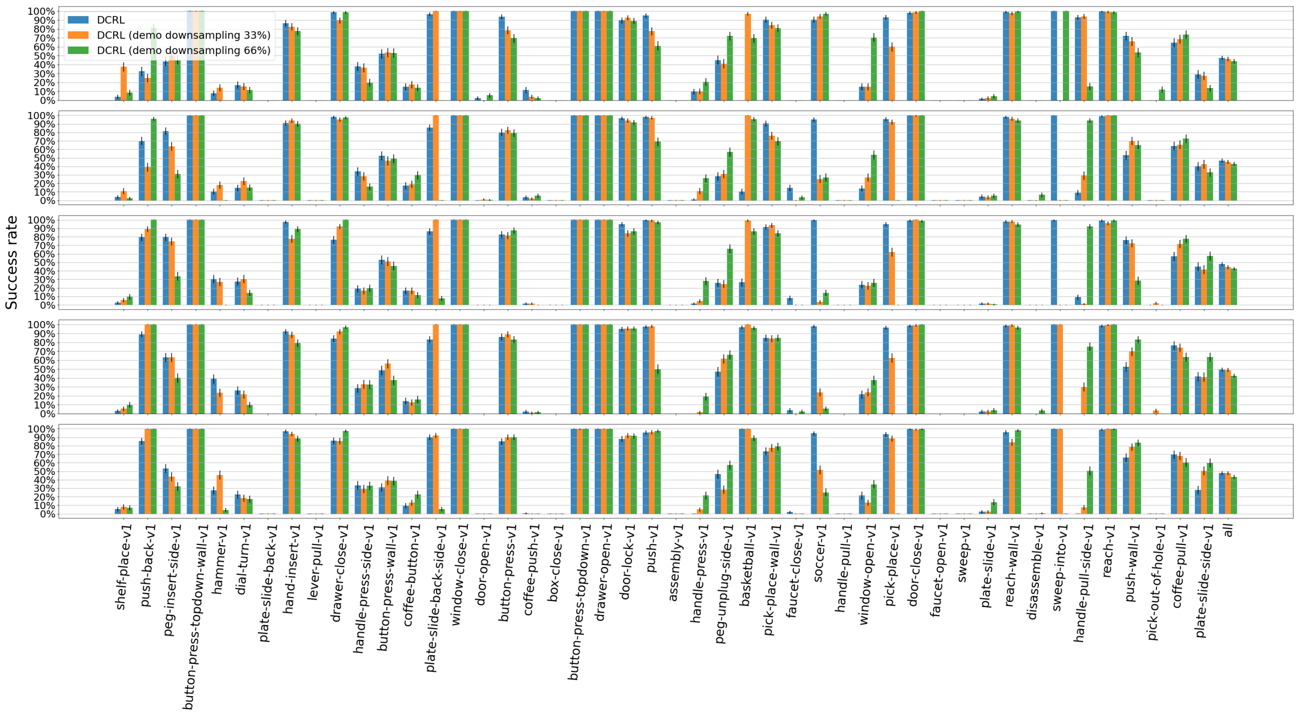


Figure 14. Success rate of DCRL using a demonstration dropout of 0%, 33% and 66%, on each Meta-World task in the few-shot experiment, for inputs consisting of one demonstration (top row) to five demonstrations (bottom row).

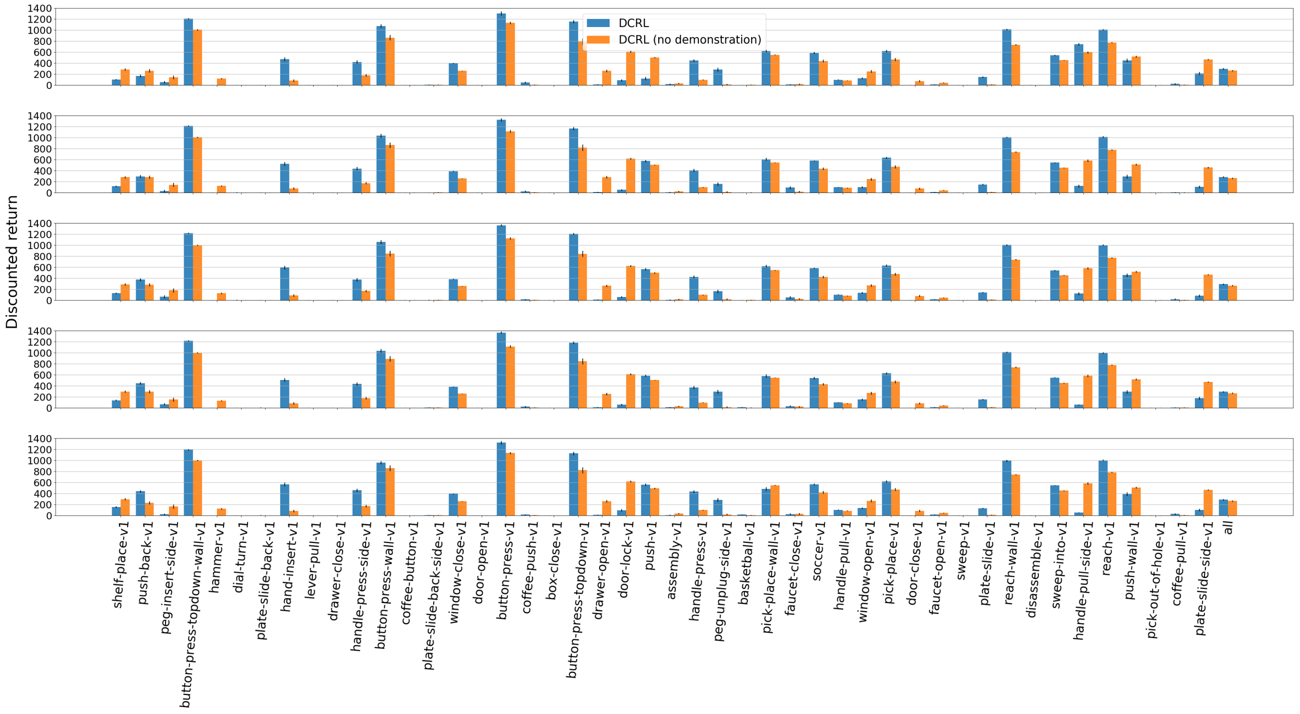


Figure 15. Discounted return of DCRL and DCRL trained without demonstration on each Meta-World task in the few-shot experiment for one demonstration (top row) to five demonstrations (bottom row).

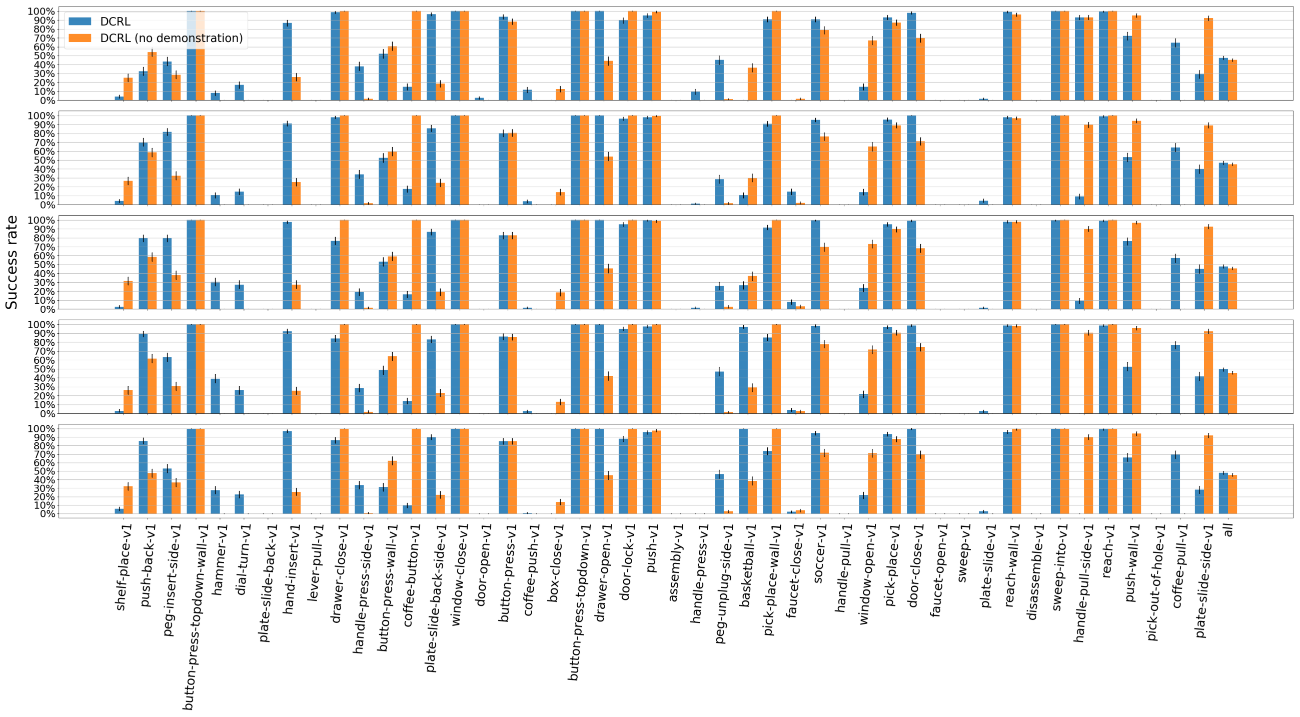


Figure 16. Success rate of DCRL and DCRL trained without demonstration on each Meta-World task in the few-shot experiment for one demonstration (top row) to five demonstrations (bottom row).

to know how important the full history is. So, Figure 17 plots the success rate when using only the last four observations, as is usually done for the Atari benchmark.

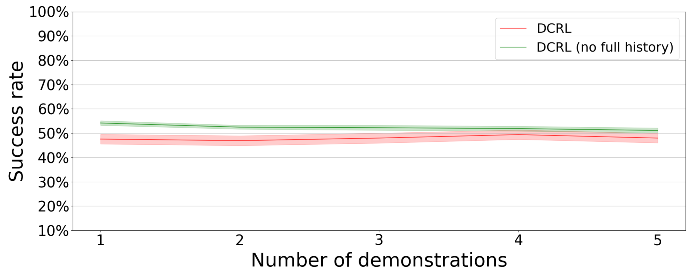


Figure 17. Success rate of DCRL and DCRL trained with only the four last observations averaged on all Meta-World tasks in the few-shot experiment.

The two approaches yield comparable results, suggesting that the last four observations are sufficient to address partial observation in the Meta-World benchmark.

4.9. Meta-IRL Baseline: Discussion

We had wished to provide results for PEMIRL (Yu et al., 2019a), as a meta-IRL baseline. This algorithm was designed under the simplifying assumption that the transition model does not vary from one task to another, and when trying to extend the algorithm to lift this assumption, we encounter a difficulty that we now discuss.

First, note that both the Meta-World and navigation benchmarks considered in our paper, have transitions that depend on the task, violating the assumptions of PEMIRL. This is because the robot will collide with different obstacles in different tasks. Moreover in Meta-World, some objects are constrained to three DOF (e.g. a plate sliding on a table), some to one rotational DOF (e.g. a door), and others to one linear DOF (e.g. a sliding window).

Now, PEMIRL is formulated in terms of a *context variable* $m \in \mathcal{M}$ that captures the difference between different tasks, which has a prior distribution $p(m)$ that determines a probability mass or density function over the set of tasks. One part of the PEMIRL model is an inference model $q_\psi(m|\tau)$ with parameters ψ, which takes an expert demonstration τ and outputs an approximate posterior over the context variable m. The idea (Yu et al., 2019a, equation 9) is to train ψ to maximize

$$\mathbb{E}_{m \sim p(m), \tau \sim p_\theta(\tau|m)}[\log q_\psi(m|\tau)],$$

where $p_\theta(\tau|m)$ is the distribution over state-action sequences $(s_1, a_1), \ldots, (s_T, a_T)$, when actions are sampled according to the optimal entropy-regularized policy for rewards $r_\theta(s_t, a_t)$ with parameters θ. To implement this, the PEMIRL meta-training procedure (Yu et al., 2019a, Lines 2–3 of Algorithm 1) involves the following steps:

Infer a batch of latent context variables from the sampled demonstrations: $m \sim q_\psi(\cdot|\tau_E)$.

Sample trajectories $\mathcal{D}$ from $\tau \sim p_\theta(\tau|m)$, with the latent context variable fixed during each rollout.

Now, if we consider $q_\psi(\cdot|\tau)$ as a distribution over a Euclidean space $\mathcal{M} = \mathbb{R}^n$, these steps require us to define a distribution $p_\theta(\tau|m)$ for *all* values of $m \in \mathbb{R}^n$, not just for those m corresponding to a set of training tasks. This in turn requires a transition distribution $\mathbb{P}(s_{t+1}|s_t, a_t, m)$ that is appropriate for all $m \in \mathbb{R}^n$. There are certainly some families of tasks for which such a definition would be natural, for instance a family where the task with parameter m involves lifting an object whose mass is m, and we have $\mathcal{M} = \mathbb{R}_{\geq 0}$. However, it is not obvious how to define an appropriate transition distribution parameterized by $m \in \mathbb{R}^n$, which includes the transition distributions of all the Meta-World tasks or the mazes in our navigation benchmark as special cases.

In future, it would be interesting to devise new meta-IRL algorithms that lift the assumption that the transition model does not vary from one task to another.

5. Navigation Benchmark

5.1. Settings

5.1.1. Maze Generation and Task Splits

We generated 60 square mazes, each of size 800×800, using an open-source VizDoom maze generator[4]. The maze generator splits the 800×800 square into 16 smaller 200×200 squares. Then, it deletes a random number of edges from these 16 squares, and checks if all points are reachable from any other point in the current maze. The algorithm continues to delete edges until this condition is satisfied. There is an additional check to ensure that the mazes generated are distinct.

The resulting mazes are shown in Figure 18. In this benchmark, we split these mazes into two groups: the training tasks use mazes MAP01 to MAP50; and the test tasks use mazes MAP51 to MAP60.

5.1.2. States, Actions and Transitions

The state is of the form $s = (s_p, s_v, s_g, s_o) \in \mathbb{R}^7$ where $s_p, s_v, s_g \in [0, 800]^2$ are the agent's position, the agent's velocity and the goal position, expressed in Cartesian coordinates, and $s_o \in [0°, 360°)$ is the agent's orientation. There are four discrete actions (forward, backward, turn left and turn right). The initial state is sampled uniformly from the state space. Note that s_p and s_g may be sampled close to each other, so bad policies can still achieve a non-zero success rate.

Transitions are generated by VizDoom. Given the current state s and action a, we denote the next state by $f_{\text{VizDoom}}(s, a)$.

5.1.3. Reward Function

Each task μ of the navigation benchmark has a different maze layout, but all tasks share the reward function

$$R_{\text{nav}}(s,a) = (\|s_p - s_g\|_2 - \|f_{\text{VizDoom}}(s,a) - s_g\|_2) - 20 \times \mathbf{1}_{\|s_p - \text{closest\_wall}(s_p)\|_2 < 20} + 100 \times \mathbf{1}_{\|s_p - s_g\|_2 < 64}.$$

The term in parentheses is the variation in the Euclidean distance from the agent to the goal. The second term is a penalty for being too close to the closest wall. The last term is a bonus for successfully reaching the goal.

5.2. Few-Shot Imitation

5.2.1. Discounted Returns and Success Rates for Each Task

Figures 19 and 20 compare the discounted return and success rate for each navigation task in the few-shot imitation experiment (Section 4.2.1 of the main paper). In contrast with the corresponding figures for the Meta-World benchmark (Figures 2 and 3), the performance of DCRL on the navigation benchmark increases with the number of input demonstrations.

5.2.2. t-SNE of Demonstration Embbedding

As we did for the Meta-World benchmark, we now explore the representations learned by DCRL using t-SNE plots. Figure 22 shows plots t-SNE$(\mathbf{d})$ of collections of demonstrations $\mathbf{d}$. As these collections consist of variable numbers of demonstrations n and time steps T, we average over these dimensions, so these plots show the result of a mapping from $\mathbb{R}^7$ to $\mathbb{R}^2$. Figure 21 shows t-SNE plots of the time-averages of a randomly-initialized embedding Φ_{random} and a learned embedding $\Phi_{\text{learned}}(\mathbf{d})$. Thus, each of these plots shows the result of a mapping from $\mathbb{R}^{64}$ to $\mathbb{R}^2$. The embedding function Φ_{learn} was learned on the training mazes, but the plots show its application to the test mazes. In both plots, different colours correspond to different test mazes.

Unlike the corresponding plots for Meta-World, the plots for averaged demonstrations t-SNE$(\mathbf{d})$ show no clustering by task, for any n. The time-averages of the learned embedding $\Phi_{\text{learned}}(\mathbf{d})$ for a single demonstration, $n = 1$, shows no clustering either. However, the test tasks form increasingly distinct clusters for Φ_{learned} as the number of demonstrations increases to $n = 4$, and they form distinct clusters for Φ_{random} even for $n = 1$. This result provides some motivation for the use of encoders with cross-demonstration attention. On the other hand, given that the test tasks are so muddled by Φ_{learned} for $n = 1$, the success rate of the few-shot policy with $n = 1$, appearing in Figure 20, might seem surprisingly high, but this no-longer remains a surprise in view of the results for a no-demonstration policy presented in Section 5.5.

Figure 23 relates the plot of t-SNE$(\Phi(\mathbf{d}))$ with $n = 4$ to the corresponding maze layouts. At the bottom of this plot, the

[4]The maze generator can be found at `https://github.com/agiantwhale/NavDoom`.

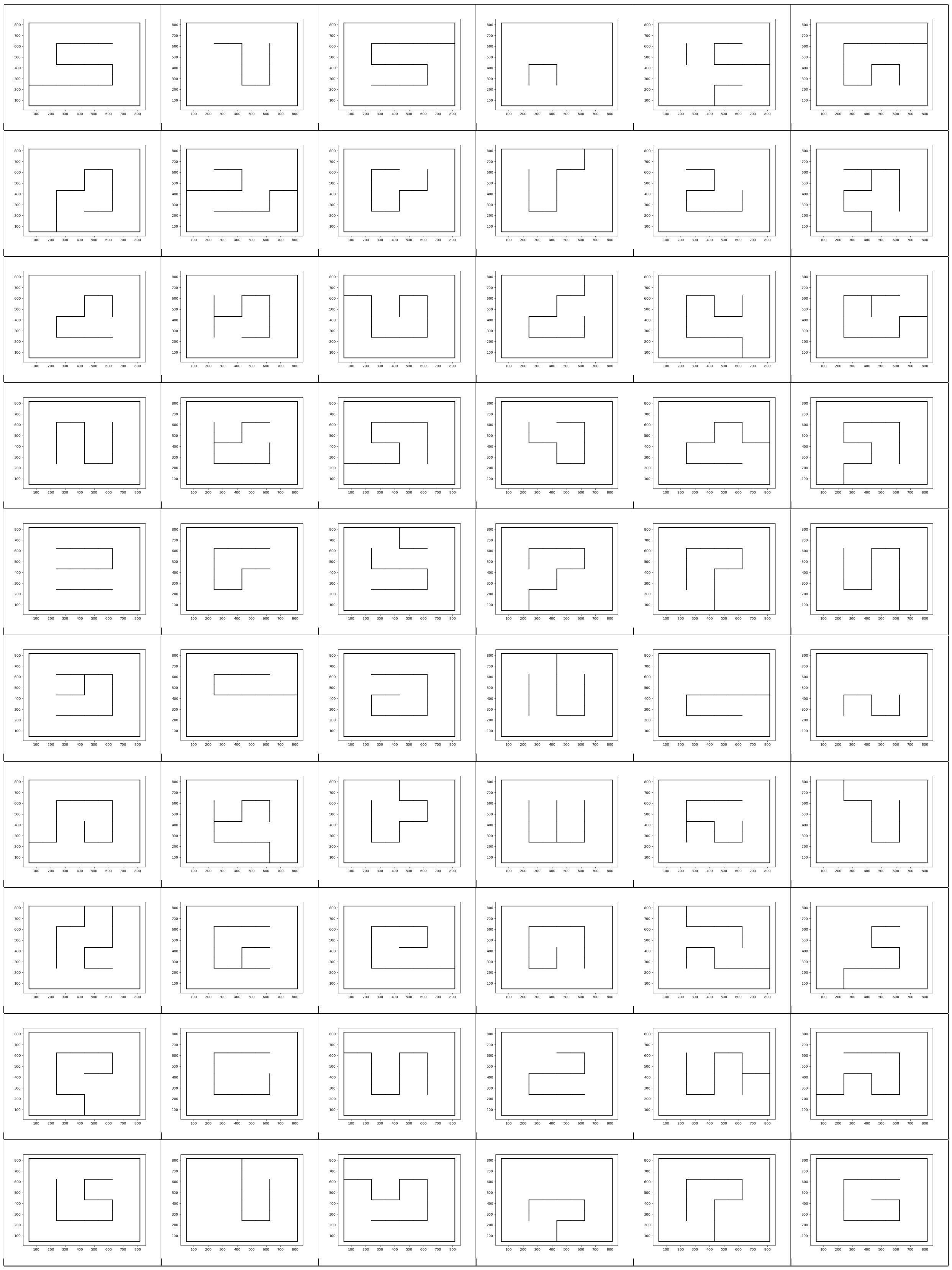


Figure 18. Layout of the 60 mazes used in the navigation benchmark, in reading order from MAP01 (top-left), to MAP06 (top-right), and on to MAP60 (bottom-right).

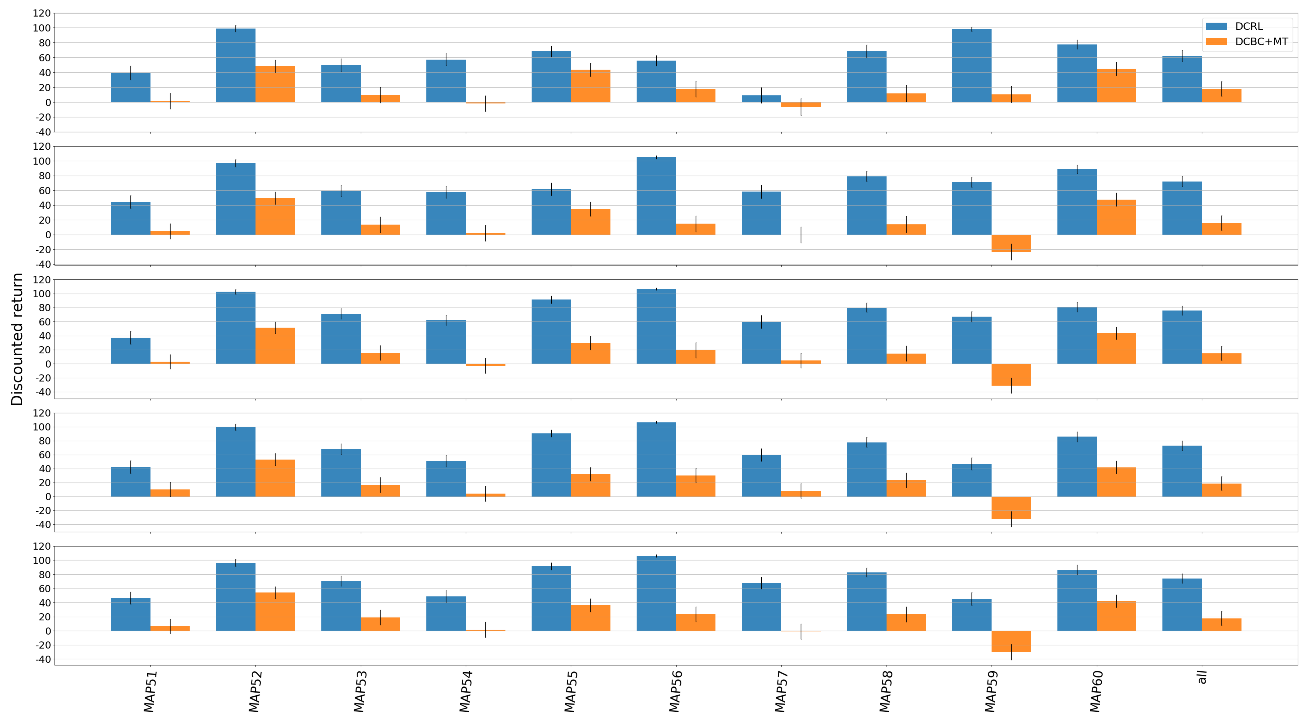


Figure 19. Discounted returns of DCRL and DCBC+MT on each navigation task in the few-shot experiment, for one demonstration (top row) to five demonstrations (bottom row).

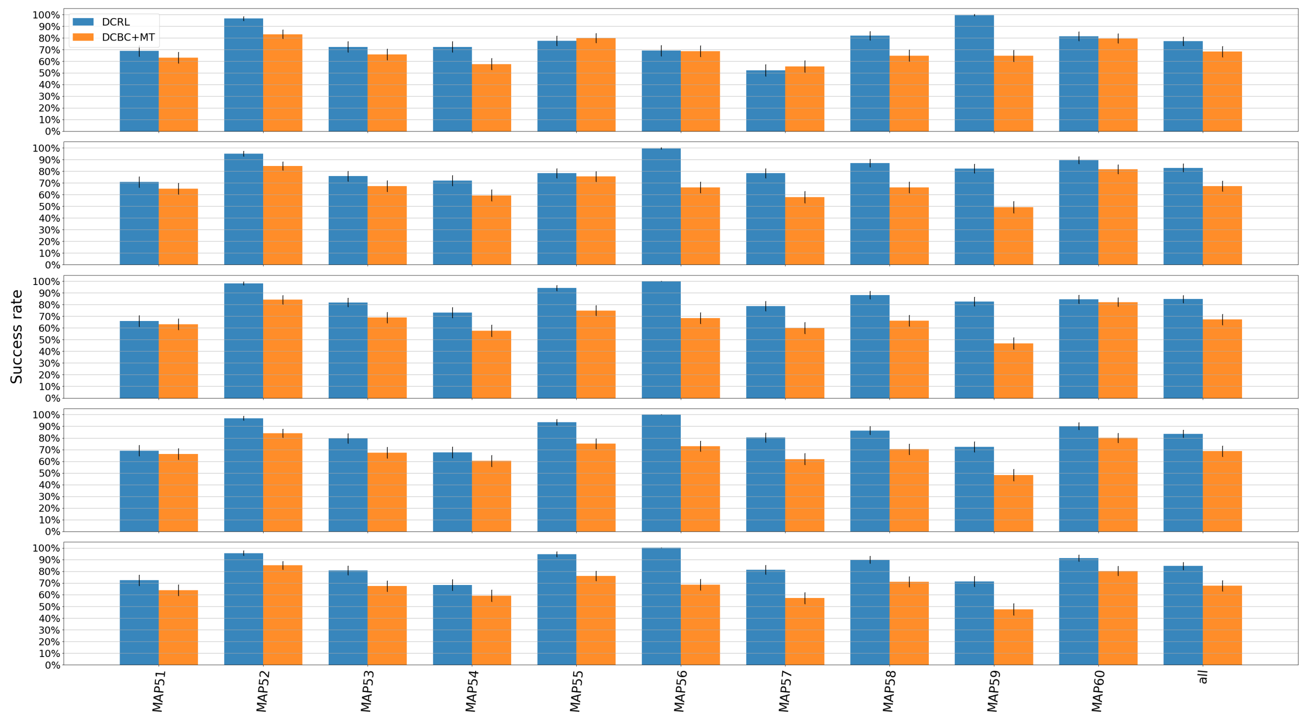


Figure 20. Success rates of DCRL and DCBC+MT on each navigation task in the few-shot experiment, for one demonstration (top row) to five demonstrations (bottom row).

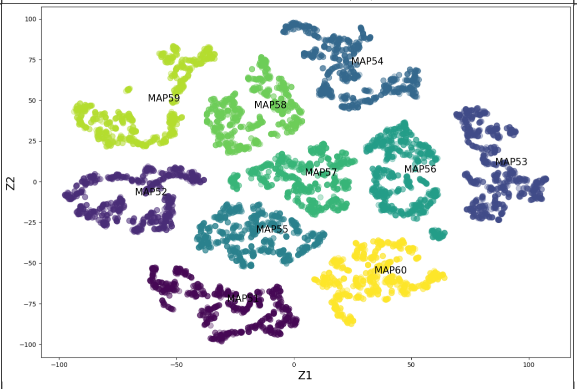
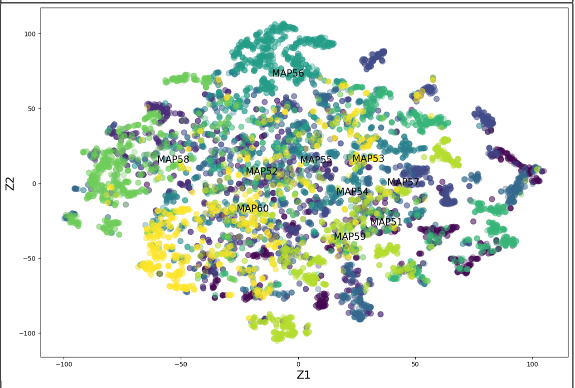
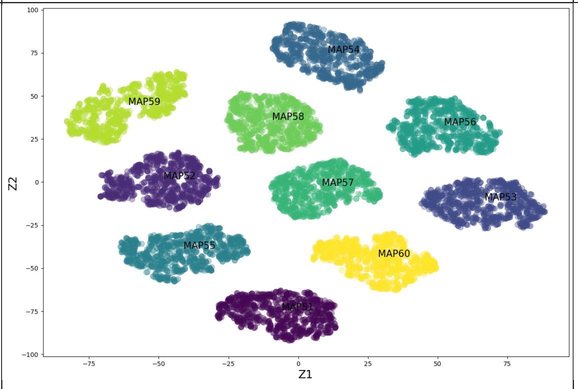
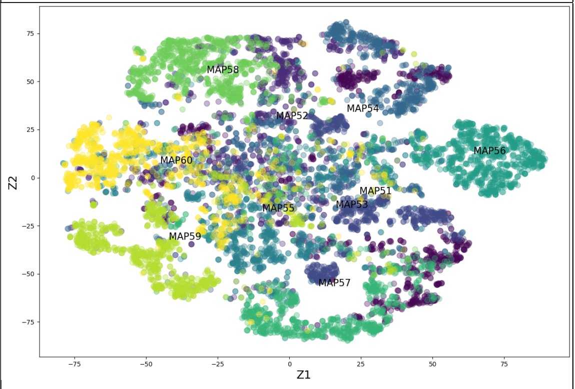
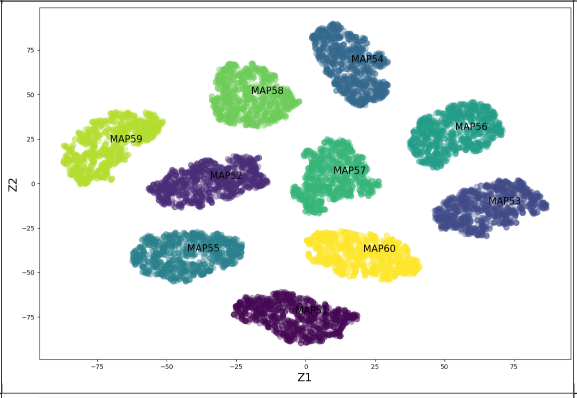
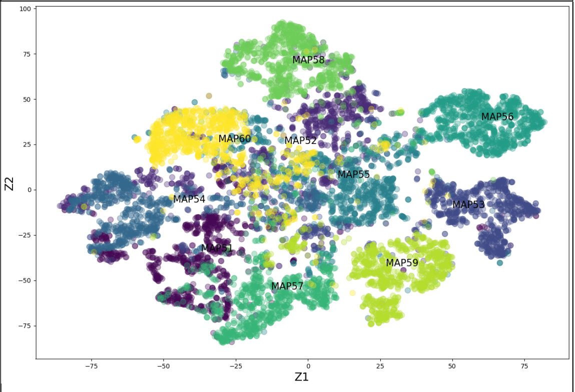
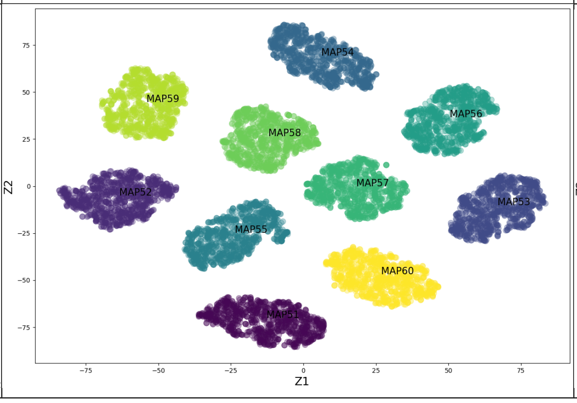
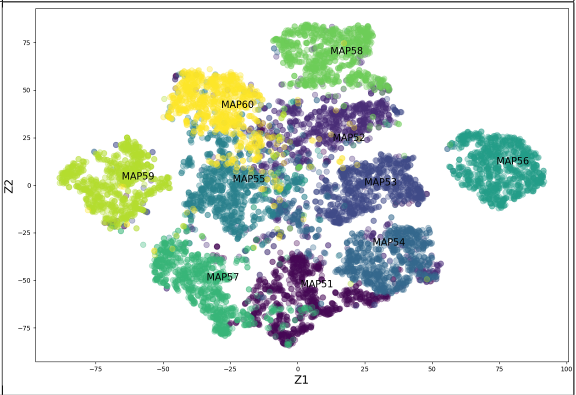
	t-SNE($\Phi_{\text{random}}(\mathbf{d})$)	t-SNE($\Phi_{\text{learned}}(\mathbf{d})$)
$n = 1$		
$n = 2$		
$n = 3$		
$n = 4$		

Figure 21. t-SNE plots of the random embedding $\Phi_{\text{random}}(\mathbf{d})$ of collections of demonstrations (left), and of the learned embedding $\Phi_{\text{learned}}(\mathbf{d})$ (right) for one (top row) to four (bottom row) demonstrations per collection.

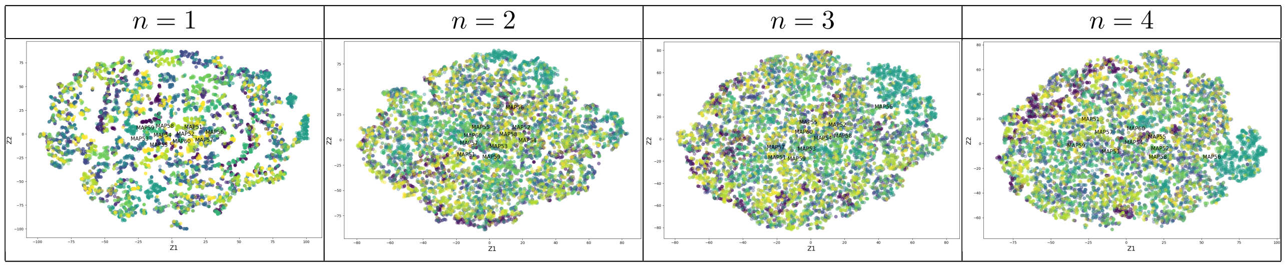


Figure 22. t-SNE plot of collections of demonstrations $\mathbf{d}$, averaged over their time and demonstration dimensions, for one (left) to four (right) demonstrations per collection.

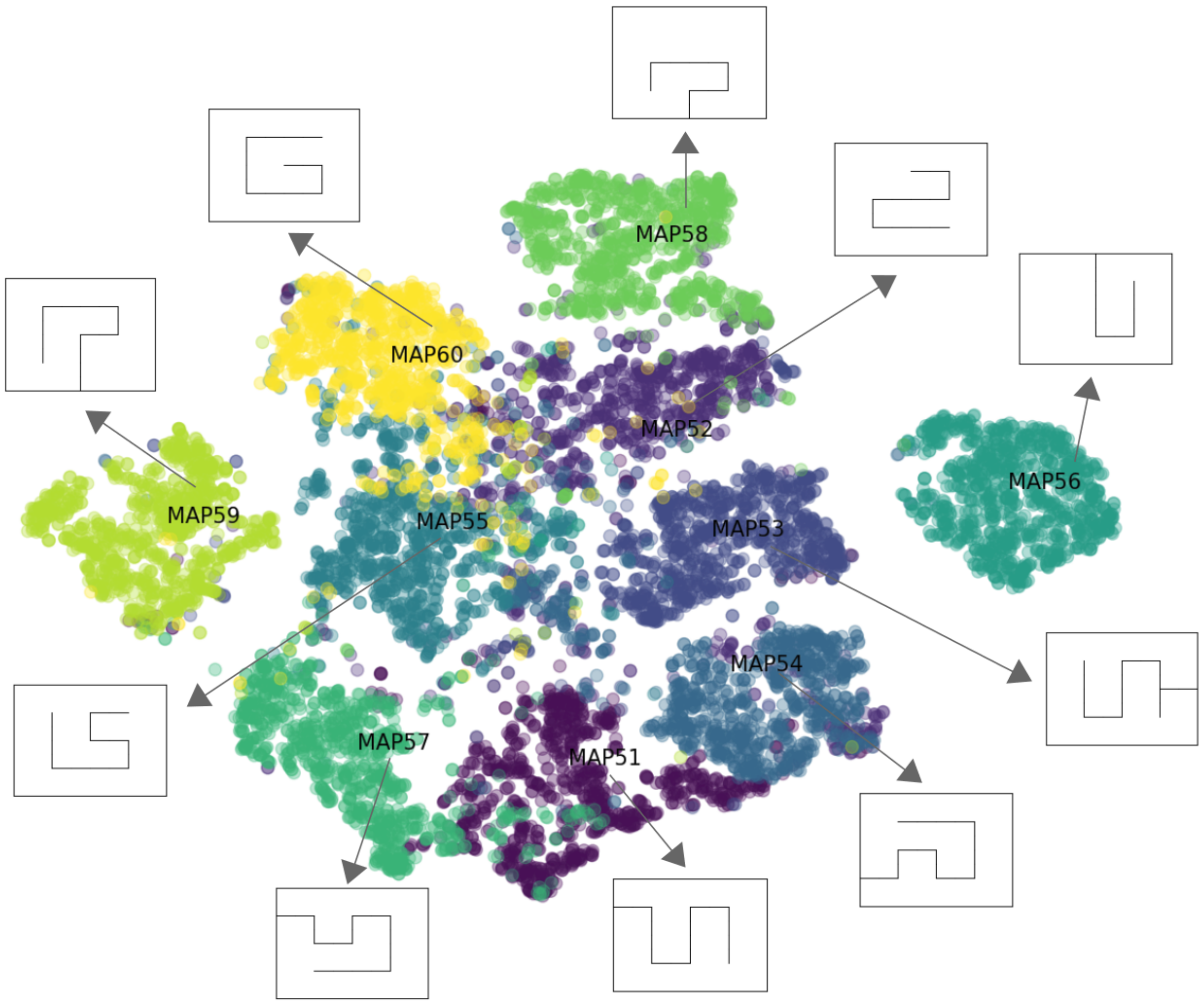


Figure 23. t-SNE plot of embedded collections of $n = 4$ demonstrations, for the 10 test mazes, using the learned embedding Φ_{learned}.

clusters for MAP54 and MAP57 overlap with that for MAP51. One might interpret this observation as a consequence of the fact that all three mazes have a wall on their left side, that connects to the outer square. Also, in the centre of the plot, the clusters for MAP55, MAP52 and MAP60 overlap. One might interpret this as a consequence of the fact that these are the only maze layouts where it is possible to walk in a loop around the outer perimeter without encountering a wall.

Although we presented results only for test tasks here, we have repeated the above t-SNE analysis, while plotting both test and training tasks at the same time. The results compellingly show that test demonstrations are mapped close to training demonstrations, when the maze layouts of those demonstrations are similar, for $n \geq 2$.

5.3. Few-Shot Imitation With Finetuning

Figures 24 and 25 show the discounted returns and success rates for the finetuning experiment (Section 4.2.2 of the main paper). We observed that DCBC+REPTILE and DCBC+MT often end up getting stuck against a wall, resulting in negative returns for some tasks.

5.4. Temporal and Cross-Demonstration Attention

Figure 26 depicts the attention of the first temporal encoder layer and the first cross-demonstration encoder layer. In this plot, points represent demonstrations steps, and the more the layer attends to a step, the darker its point is. We measure how much a layer attends to a step as the sum of the self-attention outputs over the heads of that layer.

The temporal encoder layer, which contextualises a single demonstration with itself, pays particular attention to the start and end points. While neither encoder layer pays much attention when the demonstrator moves in a straight line, the temporal layer pays close attention when the demonstrator turns, possibly because the demonstrator's turns are particularly informative about the current maze layout.

The cross-demonstration encoder layer contextualises multiple demonstrations with themselves, for each time step independently. As the input to the first such layer already combines information from multiple times, and the information is distributed across multiple heads, we think it is difficult to offer an insightful interpretation for the behaviour of this layer, seen in the bottom row of Figure 26.

One might hypothesize that the cross-demonstration layer tries to summarize demonstrations to avoid redundancy. That is, if two demonstrations take roughly the same path, then the cross-demonstration encoder will pay attention to only one of these demonstrations. For instance, we see that the rightmost demonstration of the middle maze of Figure 26 is assigned little attention.

5.5. No-Demonstration Experiment

As for Meta-World, we conducted an experiment in which DCRL was trained with $n = 0$ demonstrations as input. The architecture of this policy was identical to that used for DCRL with $n > 0$, but the demonstration input was an array of zeros. The same training-test split was used as when training DCRL with $n > 0$ demonstrations. Figures 27 and 28 show the resulting returns and success rates. The no-demonstration policy achieves a 71% success rate overall, but DCRL with $n \geq 4$ has a higher success rate for each task individually. The no-demonstration policy also attains a positive reward for all tasks, outperforming the behaviour-cloning baselines of Figure 19. However, the return of DCRL is higher than that of the no-demonstration policy, for each task individually, given $n \geq 3$ demonstrations as input.

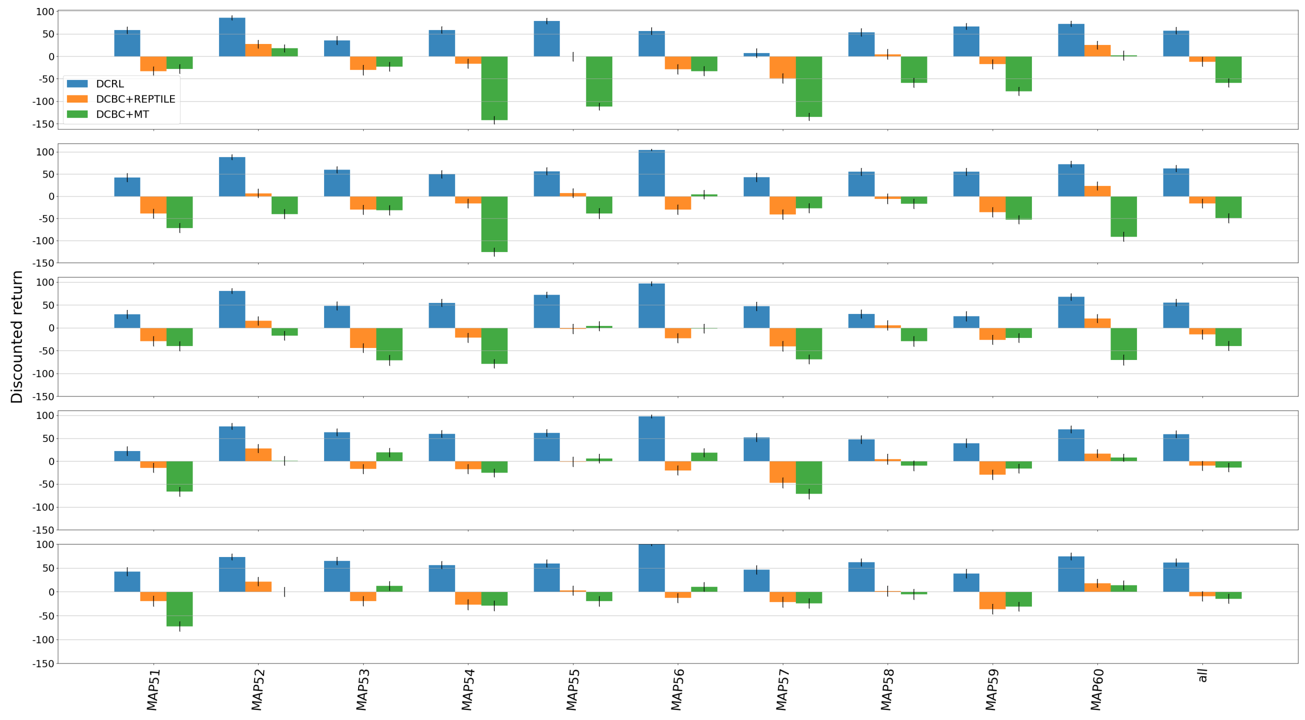


Figure 24. Discounted return of DCRL, DCBC+REPTILE and DCBC+MT on each navigation task in the few-shot experiment with finetuning for one demonstration (top row) to five demonstrations (bottom row)

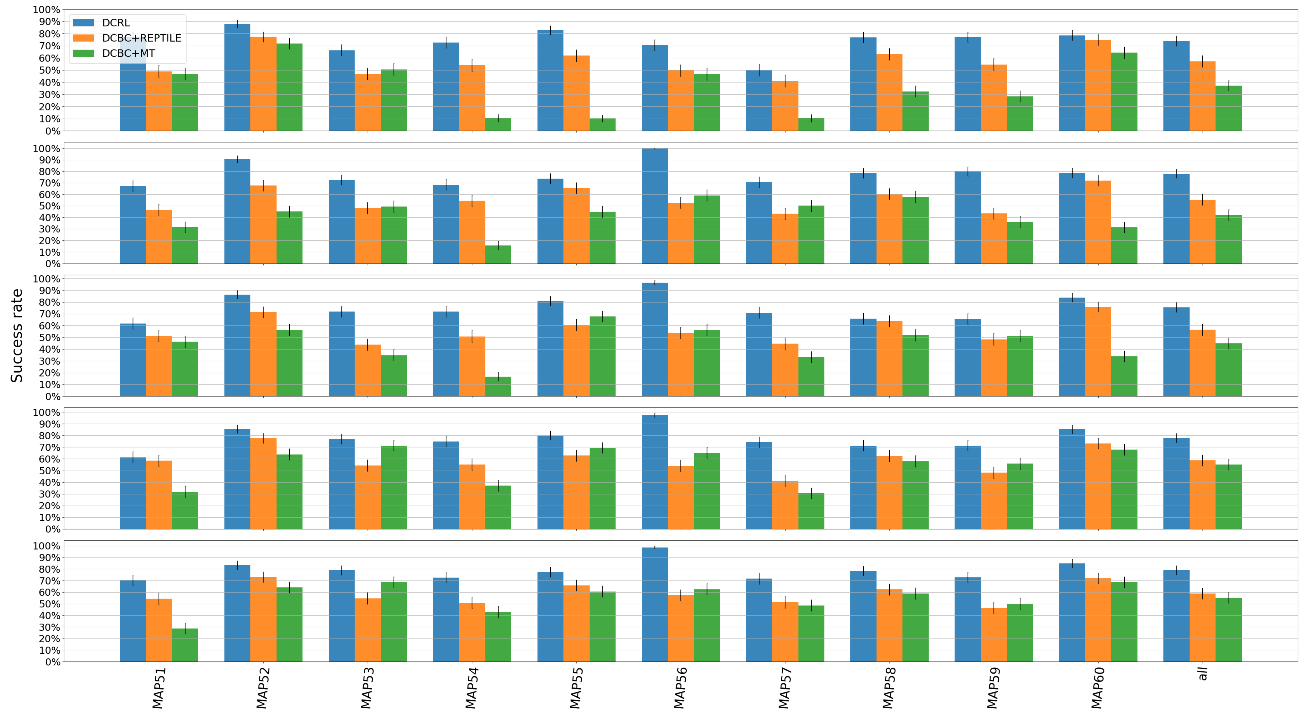


Figure 25. Success rate of DCRL, DCBC+REPTILE and DCBC+MT on each navigation task in the few-shot experiment with finetuning for one demonstration (top row) to five demonstrations (bottom row).

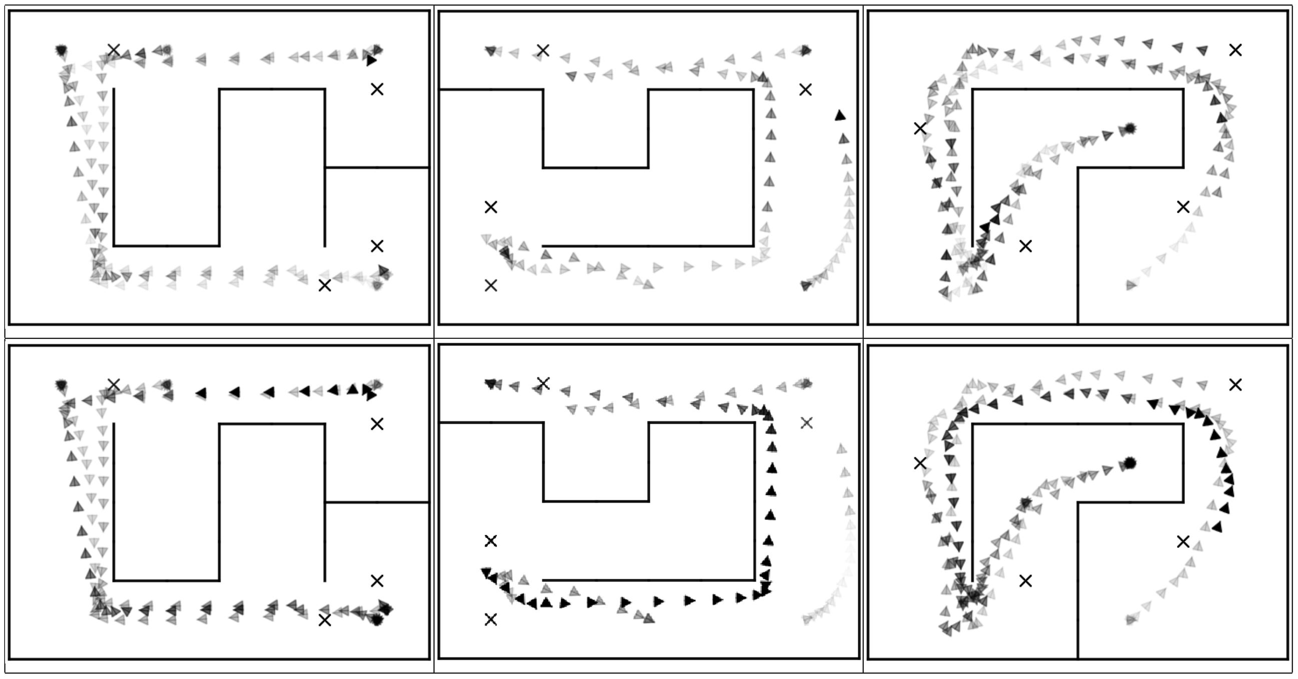


Figure 26. Attention of the first temporal encoder layer (top row) and the first cross-demonstration encoder layer (bottom row), to a collection of $n = 4$ demonstrations for three maps. The demonstrator's orientation is indicated by the orientation of the triangle. Darker triangles indicate that more attention is placed on the corresponding step. Crosses denote the goals of the demonstrations.

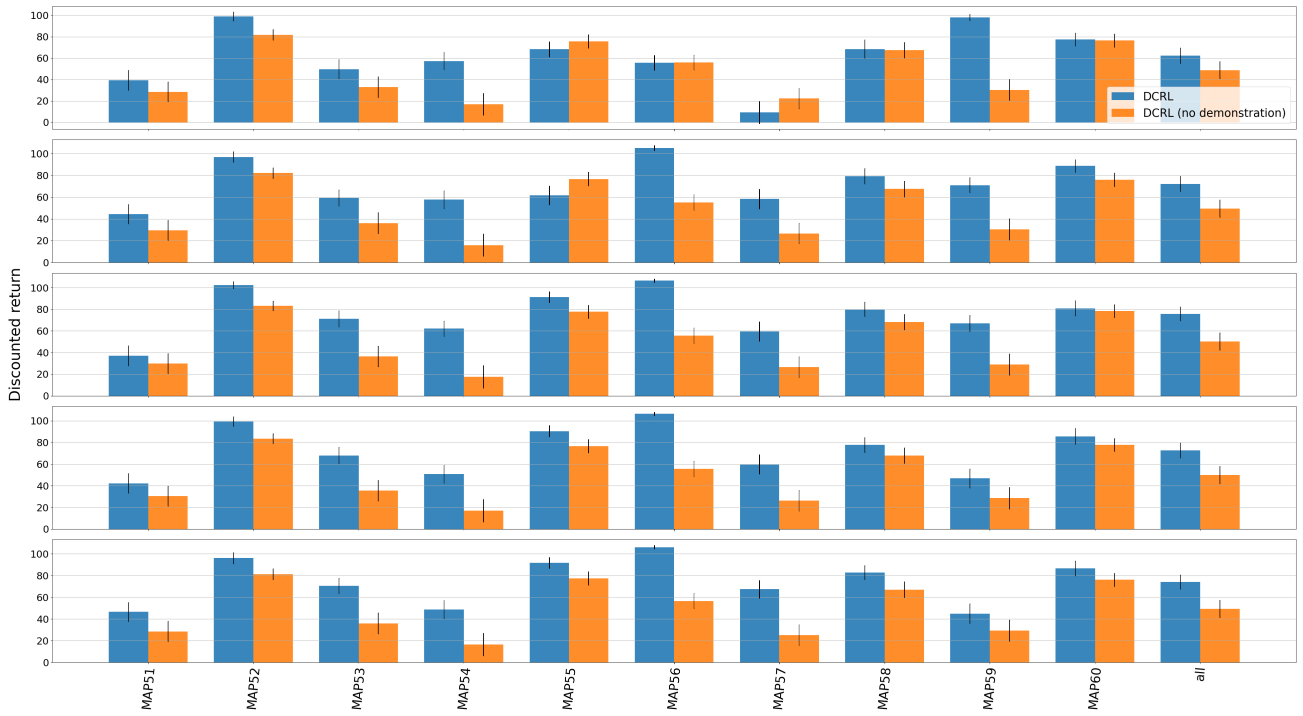


Figure 27. Discounted returns of DCRL trained with no input demonstration, for each test task of the navigation benchmark, compared with those of DCRL given one (top row) to five (bottom row) demonstrations as input.

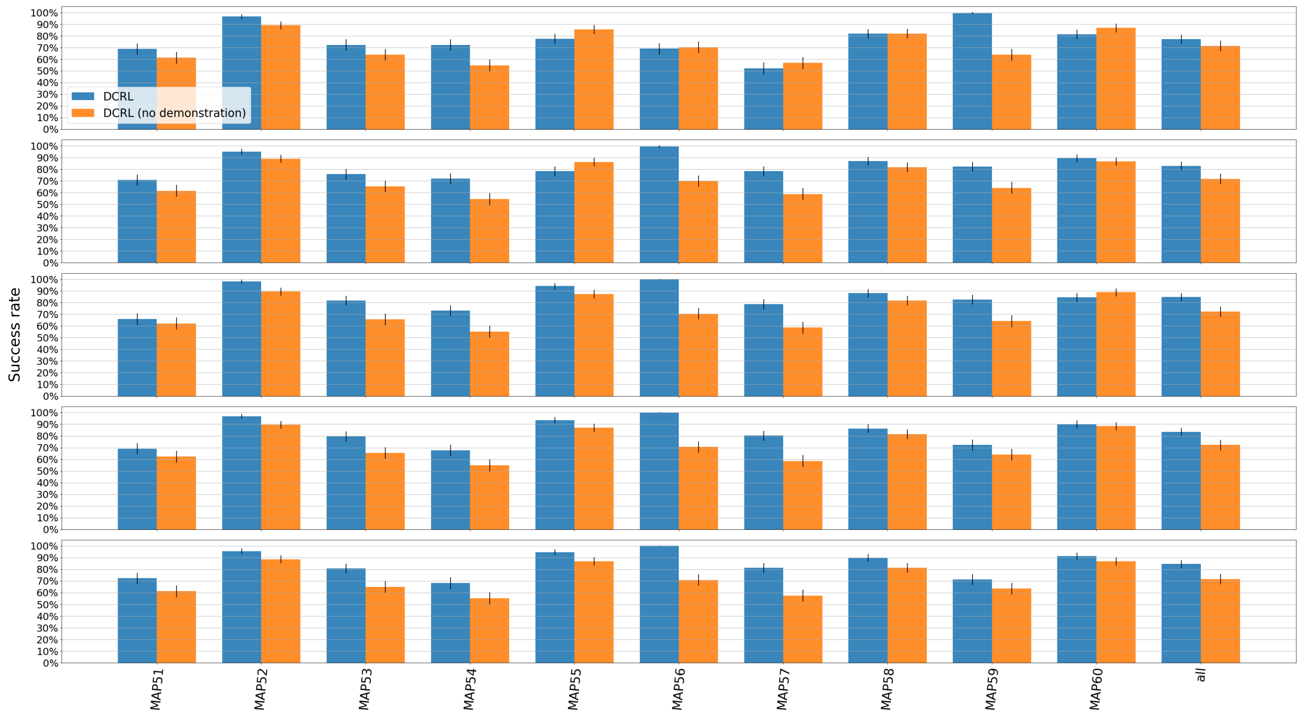


Figure 28. Success rates of DCRL trained with no input demonstration, for each test task of the navigation benchmark, compared with those of DCRL given one (top row) to five (bottom row) demonstrations as input.

5.6. No-History Experiment

Figure 29 compares policies using the full history with policies using only the last four observations. In contrast with the corresponding results for the Meta-World benchmark (Section 4.8), using the whole history performs significantly better here. This might be because the agent can use the history to avoid retracing its steps, if it gets blocked on the way to its goal.

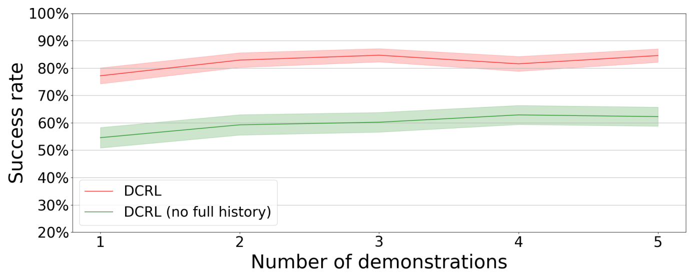


Figure 29. Success rates of DCRL with the full history and DCRL with only the four last observations, averaged over the 10 test mazes, with no finetuning.

References

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics, AISTATS*, 2010.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pp. 1026–1034, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

van der Maaten, L. and Hinton, G. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems, NIPS*, pp. 5998–6008, 2017.

Yu, L., Yu, T., Finn, C., and Ermon, S. Meta-inverse reinforcement learning with probabilistic context variables. In *Advances in Neural Information Processing Systems, NeurIPS*, pp. 11749–11760, 2019a.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning, CoRL*, 2019b.