Algorithm 1: VIGOR

Input: Contexts $\boldsymbol{c} = \boldsymbol{c}_{\text{train}} \cup \boldsymbol{c}_{\text{test}}$
Input: Mappings f_c with $\boldsymbol{O} = f_c(\boldsymbol{w})$ for each $c \in \boldsymbol{c}$
Input: Expert Descriptors $\boldsymbol{O}^{(p)} = \{\boldsymbol{O}_c^{(p)} | c \in \boldsymbol{c}_{\text{train}}\}$
Input: Initial Policies $\{q_c(\boldsymbol{w}) | c \in \boldsymbol{c}\}$
Output: Converged Policies $\{q_c(\boldsymbol{w}) | c \in \boldsymbol{c}\}$

$n_{\text{samples}} \leftarrow |\boldsymbol{O}^{(p)}|/|\boldsymbol{c}|$
$\hat{q}_c(\boldsymbol{w}) \leftarrow q_c(\boldsymbol{w}) \quad \forall c \in \boldsymbol{c}$
while *not converged* **do**
  Gather new policy samples
  $\boldsymbol{O}^{(q)} \leftarrow \{\}$
  for $c \in \boldsymbol{c}$ **do**
    $\hat{\boldsymbol{O}}_c^{(q)} \leftarrow \{\boldsymbol{w}_c^{(j)}\}_{j=1\ldots n_{\text{samples}}} \sim q_c(\boldsymbol{w})$
    $\boldsymbol{O}^{(q)} \leftarrow \boldsymbol{O}^{(q)} \cup \{f_c(\boldsymbol{w}) | \boldsymbol{w} \in \hat{\boldsymbol{O}}_c^{(q)}\}$
  end

  Train discriminator $\phi(\boldsymbol{O})$ (c.f. Eq. 3)
  $\phi(\boldsymbol{O}) \leftarrow \arg\min_{\phi(\boldsymbol{O})} \text{BCE}(\phi(\boldsymbol{O}), \boldsymbol{O}^{(p)}, \boldsymbol{O}^{(q)})$

  Update policies for each context with discriminator $\phi(\boldsymbol{O})$ (c.f. Eq. 1,)
  for $c \in \boldsymbol{c}$ **do**

$$q_c(\boldsymbol{w}) \leftarrow \arg\min_{q(\boldsymbol{w})} \mathbb{E}_{q(\boldsymbol{w},\boldsymbol{z})}[\phi(\boldsymbol{O}(\boldsymbol{w}))] + \text{KL}\big(q(\boldsymbol{z}) || \hat{q}_c(\boldsymbol{z})\big) + \mathbb{E}_{q(\boldsymbol{z})}\Big[\text{KL}\big(q(\boldsymbol{w}|\boldsymbol{z}) || \hat{q}_c(\boldsymbol{w}|\boldsymbol{z})\big)\Big]$$

    $\hat{q}_c(\boldsymbol{w}|\boldsymbol{z}) \leftarrow q(\boldsymbol{w}|\boldsymbol{z}) \quad \forall \boldsymbol{z}$
  end
end
return $\{q_c(\boldsymbol{w}) | c \in \boldsymbol{c}\}$

A Pseudocode

Pseudocode for VIGOR is provided in Algorithm 1.

B Baselines

We compare VIGOR to a number of strong IL baselines in both a trajectory-based and state-action setting.

B.1 EM+D-REX

We modify Disturbance-based Reward Extrapolation (D-REX) [7], a state-of-the-art IL algorithm to work in a trajectory-based setting and with a GMMs policy. For this, we first use EM to fit a GMM with a small number of components on each configuration in $\boldsymbol{c}_{\text{train}}$, and use the resulting distributions to draw samples for the D-REX reward. To generate the rankings required by D-REX, we multiply the covariance of each component of the resulting GMM with different scalars, resulting in GMMs with different noise levels. More precisely, we fit the EM-GMM on the samples, and multiply its component-wise covariance with a *Base Noise* σ_{base} to get an initial distribution. We repeat this process for a number of *Noise Levels*, linearly increasing the noise up to $\sigma_{\text{base}} \cdot \lambda_{\text{max}}$, where we call λ_{max} the *Noise Multiplier*. We then draw samples from each noise level of the GMMs corresponding to each context in $\boldsymbol{c}_{\text{train}}$ and rank them against each other using the comparison-based loss of D-REX, where samples drawn from a lower noise level are ranked higher. The resulting ranked demonstrations

are transformed into geometric descriptors $\boldsymbol{O}$ and used to train a reward function $R(\boldsymbol{O})$, which in turn is used by a policy optimization algorithm to optimize policies on unknown contexts $\boldsymbol{c}_{\text{test}}$. For the policy optimization algorithm, we use VIPS [14, 15]. To make results more stable, we optimize on a scaled reward $\hat{R}(\boldsymbol{O}) = \alpha_{\text{scale}} R(\boldsymbol{O})$ instead of the regular reward, where α_{scale} is a scalar *Reward Scale*. We call the resulting algorithm EM+D-REX. Comparing EM+D-REX and VIGOR, both approaches train a discriminator that uses expert demonstrations on training contexts to fit novel test contexts. Similarly, both make use of VIPS to optimize marginal policies to match multi-modal distributions over geometric behavioral descriptors. However, EM+D-REX uses the discriminator to represent a reward function, while VIGOR iteratively re-trains the discriminator until convergence of the test policies.

B.2 State-Action Baselines

We also compare our approach to common state-action based imitation learning algorithms to see how well these approaches work on versatile human demonstrations. More precisely, we compare to BC [17] and Generative Adversarial Imitation Learning (GAIL) [5] as implemented in the *Imitation* [45] repository. The state-action baselines use velocities as actions, and receive the geometric descriptors of the current robot state plus the current time-step as state information. For the Point Reaching tasks, we also experiment with encoding which of the points has been reached to make the tasks Markovian. We use the Proximal Policy Optimization (PPO) [46] implementation of Stable Baselines3 [47] as the policy for GAIL. To accommodate for versatility on an action level, we also compare to BC-GMM [13], which trains a policy whose head outputs the parameters of a GMM over actions per state. We note that we do not make use of the Low Noise Evaluation Trick proposed in Mandlekar et al. [13], which leads to minor improvements in their experiments and does not affect our results. During evaluation, we also generate each trajectory from a fixed component rather than sampling a new component mean per step for consistency with the other approaches. Experimentally, we found no significant difference in performance between these two evaluation methods. In our experiments, both BC and BC-GMM also make use of an auxiliary entropy regularization term that is similar to Zhou et al. [18]. This prevents the covariances from collapsing. All state-action baselines use regular MLPs for their policy.

B.3 Trajectory-based Baselines

Finally, we employ BC and BC-GMM on a trajectory-based level, i.e., we imitate full trajectories instead of individual state-action pairs. These baselines are added to see how simple (multi-modal) behavioral cloning works on full expert trajectories. In this setup, the methods see as input the context of the task, e.g., the position of the target points for the point reaching tasks, and output a contextual distribution $q(\boldsymbol{w}|c)$ over ProMP parameters. This distribution is implemented as a simple MLP. To discriminate between state-action and trajectory-based baselines, we append suffixes '(S)' and '(T)' respectively. For example, we denote state-action BC as 'BC(S)' and trajectory-based Mixture-Density BC as 'BC-GMM(T)'.

C Additional Task Information and Results

C.1 Planar Reacher

Setup. Each context is specified by its target positions, which are drawn from independent isotropic Gaussians with means $(0.5, 2.5)$ and $(0.5, -2.5)$ and standard deviation 0.5. Both targets have a radius of $r = 0.5$ to be considered *reached*. The robot always starts with all joints at a resting position of $0\,\text{deg}$. We collect demonstrations via a joystick-based setup to control the end-effector's (x, y)-position and rotation. This is paired with an inverse kinematics controller to control the joints.

Success Rates. We define a demonstration as successful if it reaches both target circles and ends its trajectory in the second circle, i.e., if its *target distance* is 0. As such, a demonstration is considered successful if it has the same target distance as the expert demonstrations. The left of Figure 6 shows success rates on test contexts for different approaches trained on 6 training contexts.

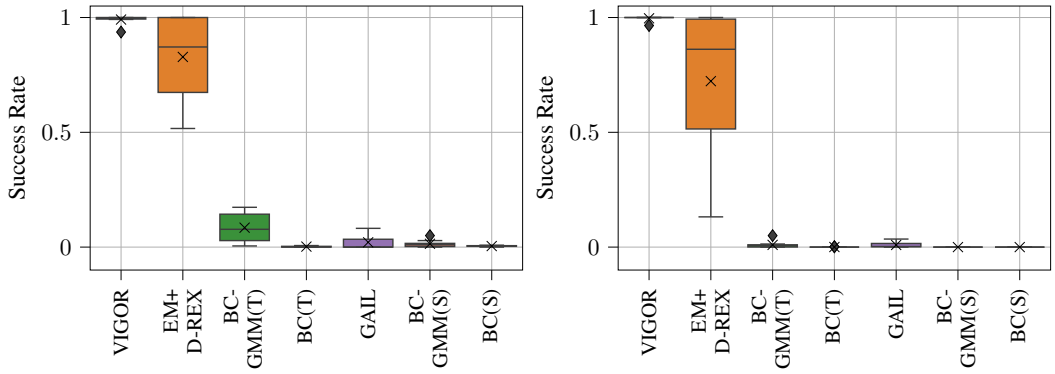

Figure 6: (Left) Mean success rates on test contexts for the Planar Reacher task. (Right) Mean success rates on test contexts for the Panda Reacher task. For both tasks, VIGOR consistently reaches both target circles, while all other methods except EM+D-REX struggle to do so. In general, the baselines with a GMM policy perform better than those without. GAIL performs the best of any state-action based approach.

Qualitative results. To explore the versatility of solutions learned by VIGOR and EM+D-REX, we visualize exemplary policies learned by both methods in Figures 7. We find that VIGOR produces versatile policies that closely match the demonstrations of the expert, with most components finding a different solution to the given task and samples of these components generally hitting both targets. At the same time, the policies of EM+D-REX often collapse to only 1 or 2 different solutions, likely due to a lack of a clear distribution matching objective.

C.2 Panda Reacher

Setup. In the teleoperation setup, the human demonstrator moves a physical robot. The robot then sends its movement to a virtual twin. The joint values of the twin are recorded over time to generate expert demonstrations. An overview of the virtual part of this setup is given in Figure 8. The environment ranges from $(0, -0.5, 0)^T$ to $(1, 0.5, 1)^T$ meters. The targets are drawn randomly from uniform distributions with range 0.1 meters and means at $(0.5, 0.2, 0.6)^T$ and $(0.3, -0.1, 0.3)^T$ meters respectively. A target is considered reached within a radius of 0.05 meters. The robot starts all trajectories at its default resting position.

We preprocess the human demonstrations by removing initial steps until the robot starts moving, and repeating the last recorded time-step before fitting the ProMPs to ensure that the fit trajectories end near the goal position. This is done to counteract a potential over-smoothing effect of the ProMPs fitting the human trajectories.

Success Rates. The right of Figure 6 shows success rates on test contexts for different approaches trained on 6 training contexts. A demonstration is considered successful if it reaches both target circles and ends its trajectory in the second circle, i.e., if its *target distance* is 0cm. This corresponds to the target distance of the expert demonstrations.

C.3 Box Pusher

Setup. Demonstrations are collected using a virtual setup similar to that of the Planar Reacher task. The demonstrator controls a mouse in the (x, y)-plane that the end-effector of the robot follows using inverse kinematics. The rotation of the end-effector is fixed to have it point straight down. The height of the end-effector is fixed such that the rod is slightly above the table. The trajectories, including the initial position of the end-effector, are fit in the (x, y) plane of the task-space for simplicity. To roll out a given trajectory on the robot, the (x, y) coordinates over time are concatenated with a fixed z-value and a downward rotation are given to the inverse kinematic controller. We sample the contexts from box translations and rotations that are feasible to demonstrate within a single smooth movement. As there tends to be a high correlation between the (x, y)-displacement and the rotation angle in the resulting contexts, we make sure that the training contexts are balanced w.r.t. this correlation.

In order for the state-action baselines to be able to choose an initial position for the trajectory, we use a separate copy of the behavioral descriptors as input for the state-action at time-step 0. In other words, we have 2 inputs in the neural network for every feature. One of these is always 0 except at step 0, where it is set to the geometric descriptors. The other is always set to the geometric descriptors, except at step 0, where it is set to 0. The output of the policy for this first time-step is then interpreted as the starting position relative to the center of the box. Outputs in later steps correspond to velocities in the (x, y)-plane.

Task visualization. Figure 9 shows final states of the Box Pusher task for VIGOR for 6 test contexts on test contexts for policies trained with demonstrations from 24 training contexts. Figure 10 shows the difference between desired and executed trajectories for 3 learned GMM component means on the same test context. These differences result from the contact with the (comparably heavy) box and are compensated by our approach.

Success Rates. The left of Figure 11 shows success rates on test contexts for different approaches trained on 6 training contexts. A demonstration is considered successful if the average distance of the corners of the final box to that of the desired final position is less than 1.5cm, which roughly corresponds to the maximum error made by any expert demonstration.

Results on training contexts. Results of VIGOR trained on 6 training contexts on the Box Pusher task for *training* contexts can be seen on the right of Figure 11.

Online task variants. To further validate the effectiveness of our geometric descriptors, we experimented with variants of BC and BC-GMM, where the distances are computed w.r.t. the current box position. We find that this choice of descriptors leads to diverging behavior on test contexts.

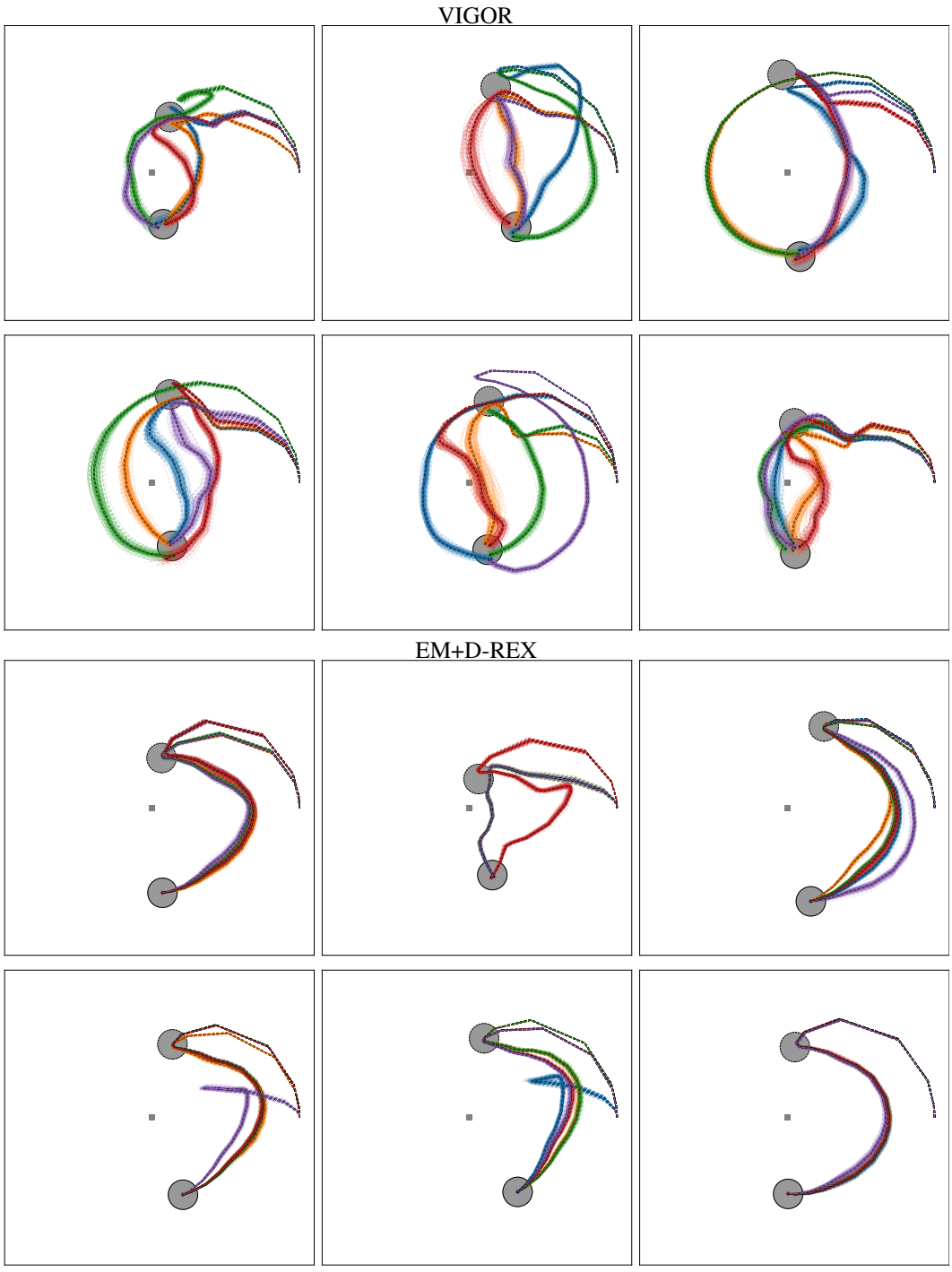

Figure 7: Visualization of end-effector traces of rollouts sampled from policies trained with VIGOR and EM+D-REX. Component means are marked with a black border, and for each component 100 samples are drawn using the same color. Note how for most contexts, VIGOR uses the components to specialize on different solutions, and how samples from this component are feasible variations of this solution. For EM+D-REX the components generally reach the targets but do not capture the versatility in the behavior.

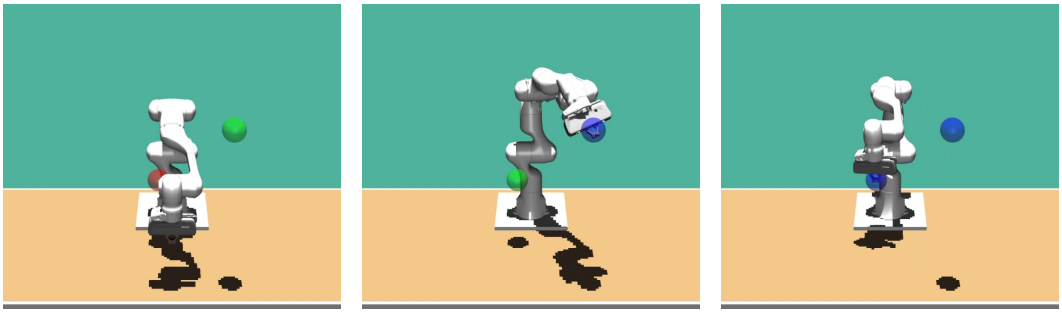

Figure 8: Virtual part of the setup for collecting human demonstrations for the Panda Reacher task. Starting from a resting position (left), the human demonstrator is tasked to reach the intermediate target (green). Once reached, both targets change color (middle). The task is successful if both targets have been reached (right).

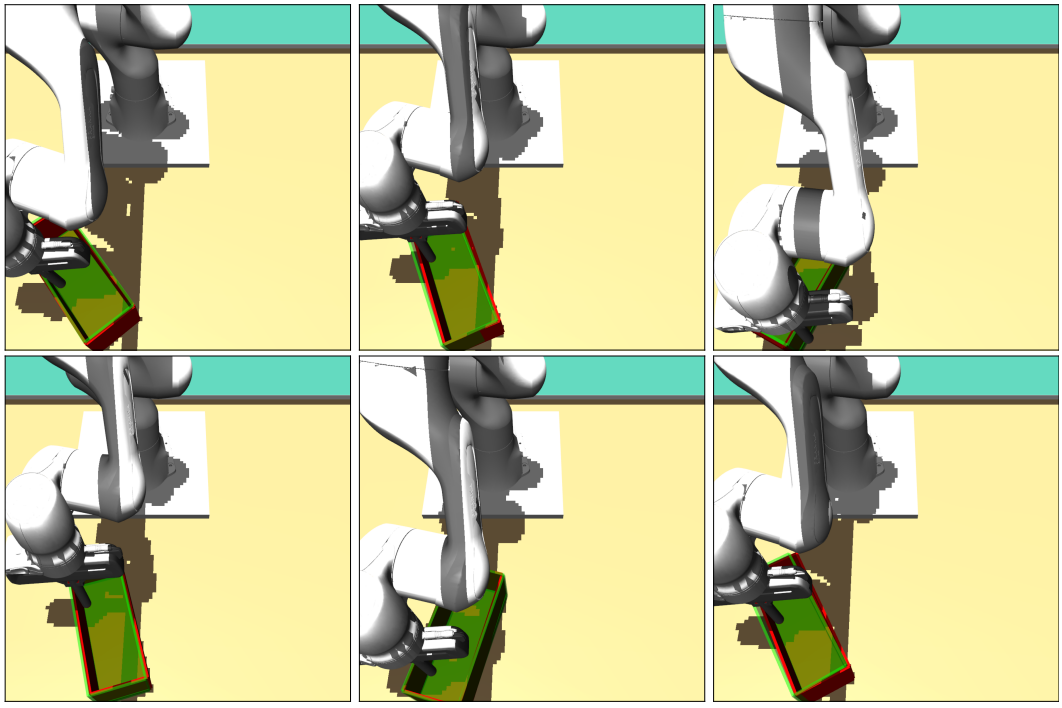

Figure 9: Visualization of the final state of the Box Pusher task on 6 test contexts for VIGOR policies trained on 24 training contexts. The pushed boxes (red) need to align as closely as possible with the target box positions (green).

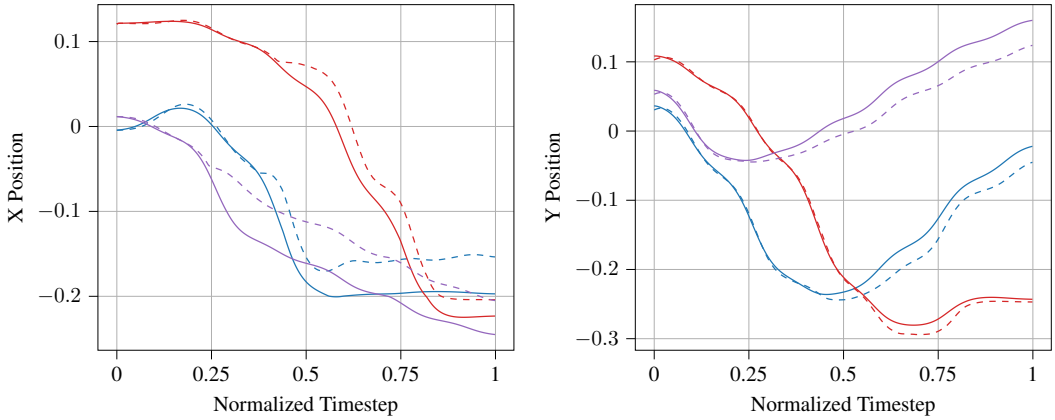

Figure 10: Difference between planned (straight line) and executed (dotted line) trajectories for 3 learned GMM component means on the same test context. The executed trajectories closely match the desired ones up to the context with the box, where the force required to push the box changes the trajectory. Note that VIGOR learns from geometric descriptors of planned expert demonstrations and is thus able to compensate for the contact with the box.

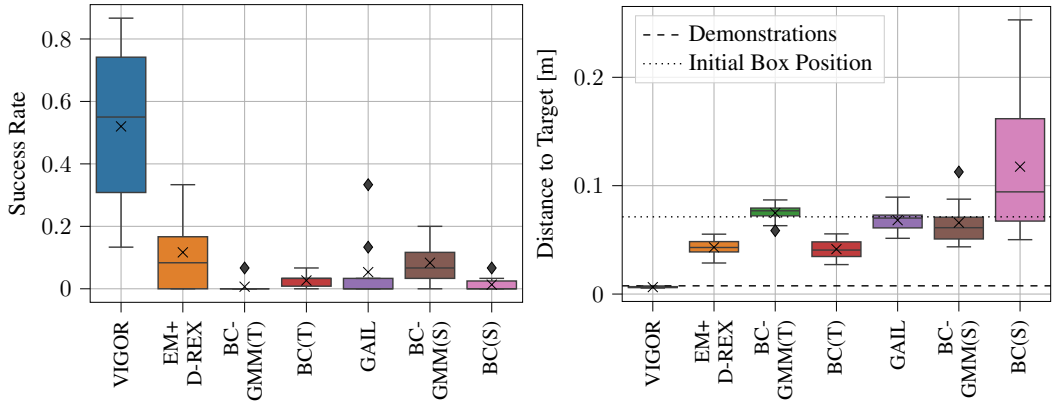

Figure 11: (Left) Mean success rates on test contexts for the Box Pusher task. We find that VIGOR produces the most successful demonstrations, while EM+D-REX and BC-GMM (S) are able to sometimes solve the task. Interestingly, BC-GMM (T) does not produce successful solutions even though it improves the distance to target, as seen on the left of Figure 4. We find that this corresponds to trajectories that push the box in the right general direction, but do not account for a precise combination of target positions and angle. (Right) Mean distance to target for training contexts of the box pushing task. Most methods reliably improve over the initial box position on training contexts. Interestingly, EM+D-REX benefits very little from evaluating on the training set, presumably because the direct correspondence to the training data is lost by the intermediate reward representation. VIGOR reaches human performance on this setup, suggesting that it can match the distribution of behavioral descriptors of the human demonstrations.

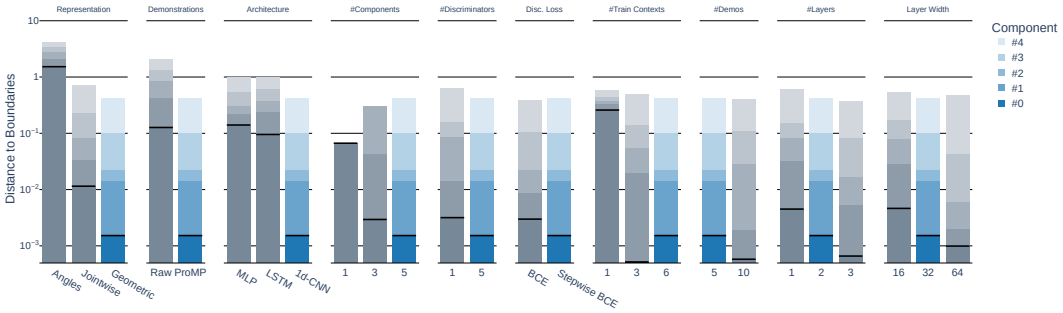

Figure 12: Full ablation study of VIGOR on the Planar Reacher task. VIGOR benefits from concise behavioral descriptors and is better suited to fit ProMP fits of the human demonstrations rather than the demonstrations themselves. We benefit from additional data in form of more demonstrations and more training contexts, as well as from additional components and discriminators.

D Ablations

We perform additional ablations for VIGOR and EM+D-REX on both the Planar Reacher and the Panda Reacher task. The ablations for VIGOR and EM+D-REX on the Planar Reacher task can be seen in Figures 12 and 13 respectively. The ablations for the Panda Reacher tasks for both methods are found in Figures 14 and 15. All figures show the performance of all mixture components of VIGOR and EM+D-REX compared to various ablations. In all figures, each bar is split into segments showing the ranked components of the learned mixture policies. In the main experiments, we evaluate the performance of the best component. This is highlighted by a black line. The blue/orange bars show the performance of VIGOR/EM+D-REX respectively, the grey bars those of the ablations.

Overall, we ablate the choice of geometric descriptors, trying both a concatenation of joint angles and context (*Angles*) as well as distances and velocity and acceleration for each robot joint (*Jointwise*). Additionally, we look at the performance of VIGOR when trained directly on human demonstrations that are sup-sampled to T time-steps (*Raw*) without first fitting a ProMP on the said demonstrations. We also evaluate the discriminator's architecture, trying out both a shared MLP per step (*MLP*) with an additional encoding of the time-step, and a Long Short-Term Memory (*LSTM*) [48]. Both architectures are configured to be of similar size to the $1d$-CNN. We also evaluate the effects of different numbers of mixture components (*#Components*), using an ensemble of discriminators (*#Discriminators*), the stepwise BCE loss (*Disc. Loss*, c.f. Eq. 3), and the amount of training contexts (*#Contexts*). Finally, we inspect the number of used demonstrations per context (*#Demos)*, and the number of $1d$-CNN layers (*#Layers*) as well as the number of convolutional channels per layer (*Layer Width*).

For EM+D-REX, we also look at how the number of fit EM components influences the performance, and how the algorithm-specific hyperparameters (see Appendix B.1) need to be tuned for good performance. The ablations show that EM+D-REX performs well if tuned right, but that it is very sensitive to the choice of hyperparameters. The most important choice of hyperparameter seems to be the number of EM components compared to the number of demonstrations per training context. Depending on the task and the remaining parameters, EM+D-REX either prefers a number of components similar to the number of demonstrations, or only a single component. We assume that this sensitivity to the choice of hyperparameters comes from the setup of EM+D-REX. While VIGOR iteratively improves its policies to match the distribution of the expert demonstrations under the geometric behavioral descriptors, EM+D-REX trains a reward function once and then uses this recovered reward function for training policies on the test contexts. As a result, any mistake in the recovered reward will directly influence the way that the newly trained policies are optimized.

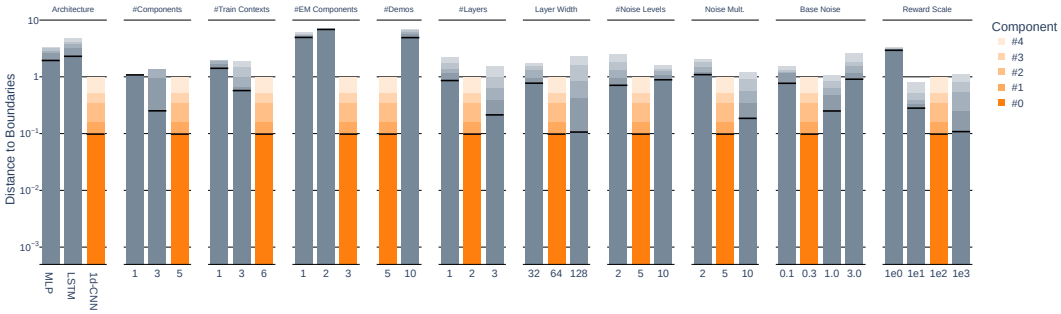

Figure 13: Ablation study of EM+D-REX on the Planar Reacher task. EM+D-REX performs well if tuned correctly, but is very sensitive to its choice of hyperparameters.

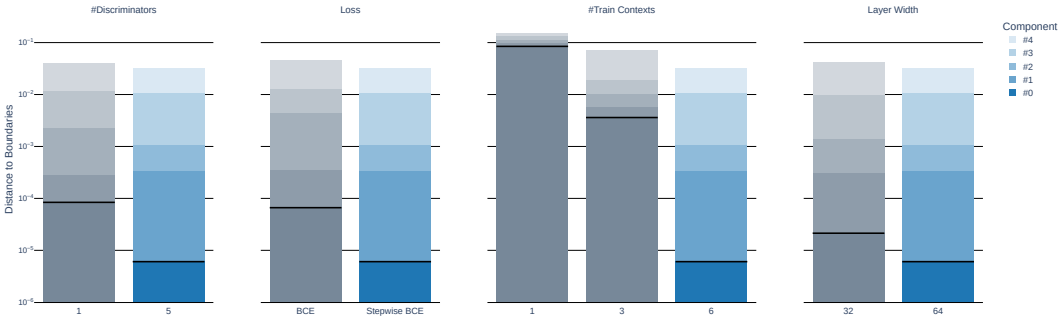

Figure 14: Ablation study of VIGOR on the Panda Reacher task.

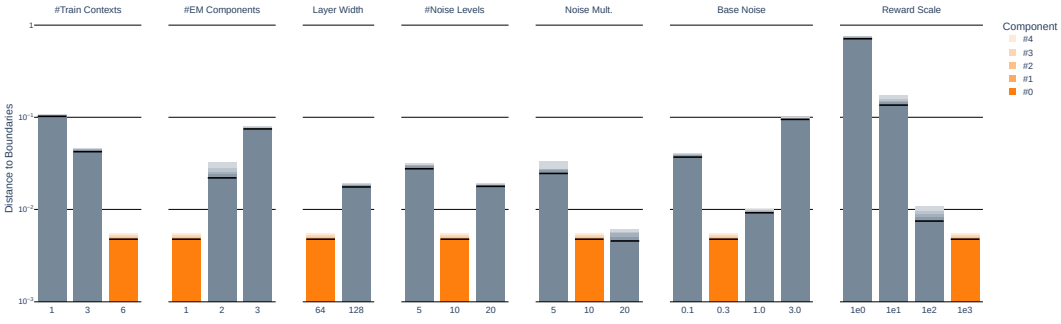

Figure 15: Ablation study of EM+D-REX on the Panda Reacher task. On this task, EM+D-REX often collapses to a single solution for the task, which can be seen by the proximity of all 5 components in each bar. The method is again very susceptible to the choice of hyperparameters, which need to be tuned on a by-task basis.

E Hyperparameters and training details

E.1 Environment Parameters

Table 1 gives an overview of default environment parameters. Note that parameters with a '*' are varied in the ablations. For each environment, we randomly draw both training and test contexts at runtime from a fixed set of contexts, making sure that training and test contexts are disjoint for any given seed.

Table 1: Default parameters of the different environments.

Parameter	Planar Reacher	Panda Reacher	Box Pusher
Trajectory length (T)	30	50	50
ProMP basis functions	5	8	7
Action dimension	5	6	2
Context dimension	4	6	3
Dimension of geometric descriptors	4	4	19
Evaluation samples per component	100	100	5
Number of training contexts ($\lvert\boldsymbol{c}_{\text{train}}\rvert$)	6	6	6
Training Demonstrations per context	5	5	3
Evaluation samples per component	100	100	5

E.2 Algorithm Hyperparameters

All neural networks are trained in PyTorch [49] using the Adam [50] optimizer. We employ Dropout [51], early stopping and Batch Normalization [52] to avoid overfitting. All methods are trained until convergence. All mixture policies use 5 components for all experiments.

All methods use the geometric descriptors of the respective environments as input. We use a learning rate of $3.0e-4$ for all methods except EM+D-REX, for which we found a learning rate of $3.0e-5$ to be more stable. VIGOR and EM+D-REX both use the equations introduced in [16] for updating their policies. We found that the method performed very similarly for a range of tested KL-Bound hyperparameters, and thus set it to 0.2 for all experiments for simplicity. We also always draw a sufficient amount of samples to estimate a full-rank quadratic surrogate to update the parameters of their mixture models in each iteration. Both VIGOR and EM+D-REX also make use of an ensemble with 5 networks, Dropout of 0.2 and use early stopping with a validation split of 0.1. We do not train the categorical distribution for either VIGOR or EM+D-REX. We find that Batch Normalization improves performance for EM+D-REX, but not for VIGOR, and as such only use it for EM+D-REX. Table 2 lists the remaining hyperparameters by task for VIGOR, Table 3 those of EM+D-REX.

For both BC and BC-GMM we find that training for 3000 and 30000 epochs on the state-action and trajectory-based settings works best respectively. Both use an entropy regularization term with a weight of $1.0e-3$. For BC-GMM, we additionally add an option to either train or not train the categorical distribution of the mixture to our hyperparameter search. If not trained, the distribution defaults to a uniform distribution over all components, similar to that of VIGOR and EM+D-REX. Remaining hyperparameters by task for BC(S) and BC-GMM(S) are given in Tables 4 and 5 respectively. Those of BC(T) and BC-GMM(T) are given in Tables 6 and 7. Hyperparameters for GAIL are listed in Table 8.

Table 2: Hyperparameters for VIGOR. Parameters with a '*' were optimized using grid searches.

Parameter	Planar Reacher	Panda Reacher	Box Pusher
$1d$-CNN layers *	2	3	4
Neurons per layer *	32	64	128
CNN kernel size	5	7	7
Batch size	64	64	64

Table 3: Hyperparameters for EM+D-REX. Parameters with a '*' were optimized using grid searches.

Parameter	Planar Reacher	Panda Reacher	Box Pusher
$1d$-CNN layers *	2	2	4
Neurons per layer *	64	64	64
CNN kernel size	5	7	7
Batch size	128	128	128
Fit EM Components *	3	1	1
Total EM Samples	8192	16384	16384
Base Noise *	0.3	0.3	0.3
Noise Levels *	5	10	5
Maximum Noise Multiplier *	5	10	2
Reward Scale *	100	1000	1000

Table 4: Hyperparameters for state-action-based Behavioral Cloning (BC(S)). Parameters with a '*' were optimized using grid searches.

Parameter	Planar Reacher	Panda Reacher	Box Pusher
MLP layers *	2	2	3
Neurons per layer *	64	32	64
Batch size *	128	64	64
Include target encoding	False	True	–
State-action framestacks *	1	1	1

Table 5: Hyperparameters for state-action-based Behavioral Cloning with a Gaussian Mixture Model policy (BC-GMM(S)). Parameters with a '*' were optimized using grid searches.

Parameter	Planar Reacher	Panda Reacher	Box Pusher
MLP layers *	2	2	3
Neurons per layer *	64	64	64
Batch size *	64	64	64
Include target encoding	False	True	–
State-action framestacks *	5	1	1
Train categorical distribution *	True	True	False

Table 6: Hyperparameters for trajectory-based Behavioral Cloning (BC(T)). Parameters with a '*' were optimized using grid searches.

Parameter	Planar Reacher	Panda Reacher	Box Pusher
MLP layers *	4	4	3
Neurons per layer *	256	64	128
Batch size *	4	4	4

Table 7: Hyperparameters for trajectory-based Behavioral Cloning with a Gaussian Mixture Model policy (BC-GMM(T)). Parameters with a '*' were optimized using grid searches.

Parameter	Planar Reacher	Panda Reacher	Box Pusher
MLP layers *	4	3	3
Neurons per layer *	64	64	128
Batch size	4	4	4
Train categorical distribution *	False	False	False

Table 8: Hyperparameters for GAIL. Parameters with a '*' were optimized using grid searches.

Parameter	Planar Reacher	Panda Reacher	Box Pusher
Discriminator MLP layers *	2	2	3
Discriminator neurons per layer *	32	64	64
Policy MLP layers	2	2	2
Policy MLP neurons per layer	32	32	32
Batch size (Discriminator) *	128	64	64
Batch size (Policy) *	128	64	256
Total time-steps *	$1e6$	$3e6$	$1e6$
Include target encoding	True	False	–
State-action framestacks *	5	5	1
Share networks	False	False	False
Policy steps	2048	2048	2048