Appendix

A Additional results

A.1 Model-based Reinforcement Learning

In model-based reinforcement learning, the key distinctions between DiffTOP and TD-MPC (Hansen et al., 2022) are: 1) TD-MPC employs the Model Predictive Path Integral (MPPI (Williams et al., 2015)) in the planning stage, whereas we utilize trajectory optimization. 2) In addition to the original loss used in TD-MPC, we use an additional policy gradient loss and back-propagate it through the differentiable trajectory optimization process to update the model parameters. Figure 6 shows that the improvement of DiffTOP over TD-MPC comes from the addition of the policy gradient loss, instead of purely changing MPPI to trajectory optimization. To be more specific, we compare TD-MPC with DiffTOP (w/o backward), a variant of DiffTOP that removes the policy gradient loss for updating the model parameters. The results indicate that TD-MPC and the DiffTOP (w/o backward) variant perform comparably, suggesting that using MPPI or trajectory optimization at test-time for action generation have similar performances. With the inclusion of the policy gradient loss, DiffTOPsignificantly outperforms both TD-MPC and the DiffTOP (w/o backward) variant, demonstrating the efficacy of adding the policy gradient loss in DiffTOP.

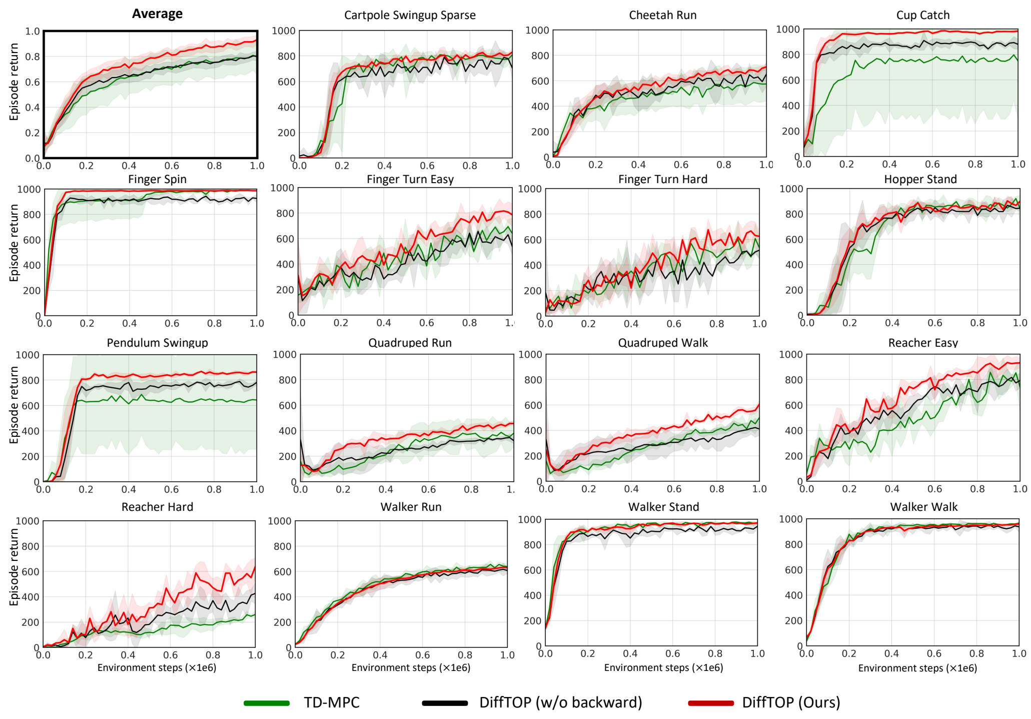

Figure 6: Performance of DiffTOP, in comparison to TD-MPC and DiffTOP (w/o backward) on 15 tasks from DeepMind control suite.

In addition to comparing the sample efficiency of DiffTOP to prior methods, we also compare the computational efficiency of DiffTOP versus TD-MPC on some of the environments. This is shown in Figure 7, where the y-axis is the return, and the x-axis is the wall-clock time used to train DiffTOP and TD-MPC for 1M environment steps. As shown, it takes more wall-clock time for DiffTOP to finish the training. In terms of computational efficiency, the results are environment-dependent. DiffTOP achieves better computational efficiency on reacher-hard and cup-catch. On pendum-swingup, TD-MPC converges to a sub-optimal value in the early training stage and DiffTOP outperforms it within 24 hours of training time. DiffTOP has similar computational efficiency on cartpole-swingup-sparse, reacher-easy, and finger-spin, and slightly worse computational efficiency

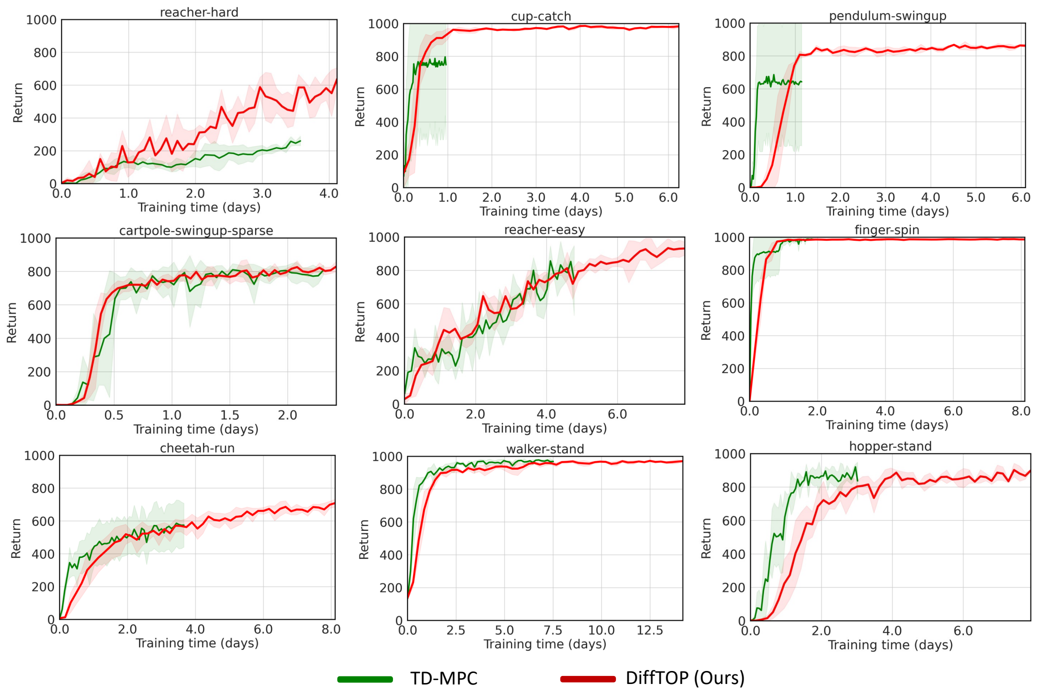

Figure 7: Return vs wall-clock time of DiffTOP and TD-MPC on some of the RL environments. The x-axis is the training time in days (24 hours), and the y-axis is the return. Both methods are trained for 1M environments steps. The training takes a long time (a few days on some environments) because the policy observation is high-dimensional images.

on cheetah-run and walker-stand compared to TD-MPC. The gap is larger on hopper-stand. The major reason for DiffTOP to take longer time for training is that solving and back-propagating through the trajectory optimization problem in Equation 4 is slower than doing MPPI as used in TD-MPC. As a reference, to infer the action at one time step, it takes 0.052 second to use Theseus to solve and differentiate through the trajectory optimization problem in Equation 4, and 0.0092 second for using MPPI in TD-MPC. However, we also want to note that the community is actively developing better and faster algorithms/software libraries for differentiable trajectory optimization, which could improve the computation efficiency of DiffTOP. For example, in all our experiments, we used the default CPU-based solver in Theseus. Theseus also provides a more advanced solver named BaSpaCho, which is a batched sparse Cholesky solver with GPU support. When we switch from the default CPU-based solver to BaSpaCho, the time cost of solving the trajectory optimization problem in Equation 4 is reduced by 22% from 0.052 second to 0.041 second. With better libraries/algorithms in the future for differentiable trajectory optimization, we believe the computational efficiency of DiffTOP would further improve.

A.2 Imitation Learning

We also present results of DiffTOP with zero initialization or random initialization, where instead of initializing the action from a base policy, the action is initialized to be 0, or randomly sampled from $\mathcal{N}(0, 1)$, on RoboMimic and Maniskill.

The results on RoboMimic is shown in Table 3. We notice a drop in performance of DiffTOP with zero or randomly-initialized actions, possibly due to the convergence to bad local minima during nonlinear trajectory optimization without a good action initialization. We note this would not be a drawback of applying DiffTOP in practice for imitation learning: one could always first learn a base policy using any behavior cloning algorithm, and then use DiffTOP to further refine the actions.

The results on Maniskill is shown in Table 4. Again, if we use zero or random action initialization with DiffTOP, the performance drops to be similar to or slightly worse than vanilla BC. Therefore, we think a good practice of using DiffTOP for imitation learning would be to always try to provide it with a good action initialization, e.g., by first training a BC policy and use its action as the initialization in DiffTOP.

	IBC	BC-RNN	Residual +BC-RNN	DiffTOP (Ours) + BC-RNN	Diffusion	IBC + Diffusion	Residual + Diffusion	DiffTOP (Ours) + Diffusion	DiffTOP (Ours) + zero init.	DiffTOP (Ours) + random init.
Square	0.04±0.00	0.82±0.00	0.84±0.01	0.90±0.02	0.88±0.03	0.68±0.05	0.88±0.02	**0.92**±0.01	0.84±0.02	0.80±0.00
Transport	0.00±0.00	0.72±0.03	0.74±0.03	0.83±0.02	0.93±0.04	0.08±0.03	0.92±0.01	**0.96**±0.01	0.42±0.01	0.36±0.04
ToolHang	0.00±0.00	0.67±0.04	0.72±0.03	0.82±0.00	0.90±0.00	0.06±0.01	0.90±0.00	**0.92**±0.01	0.00±0.00	0.00±0.00
Push-T	0.11±0.01	0.70±0.02	0.72±0.02	0.75±0.02	**0.91**±0.00	0.08±0.01	**0.91**±0.00	**0.91**±0.01	0.62±0.04	0.57±0.02

Table 3: Comparison of DiffTOP with all other mehtods on the Robomimic tasks. DiffTOP achieves the best performances on all tasks when using diffusion policy as the base policy. If zero or random initialization are used in DiffTOP, the performance drops, possibly due to the convergence to bad local minima during nonlinear trajectory optimization without a good action initialization.

	PickCube	Fill	Hang	Excavate	Pour	OpenCabinet Drawer	OpenCabinet Door	PushChair	MoveBucket
BC	0.19±0.03	0.72±0.04	0.76±0.02	0.25±0.02	0.13±0.01	0.47±0.03	0.35±0.04	0.12±0.01	0.10±0.01
BC + residual	0.21±0.04	0.75±0.02	0.75±0.02	0.27±0.03	0.12±0.01	0.49±0.02	0.36±0.03	0.15±0.02	0.10±0.01
DiffTOP(Ours) + BC	**0.32**±0.02	**0.82**±0.01	**0.85**±0.03	**0.29**±0.01	**0.17**±0.02	**0.53**±0.02	**0.45**±0.02	**0.20**±0.02	**0.15**±0.02
DiffTOP (Ours) + zero init.	0.20±0.03	0.76±0.03	0.72±0.02	0.25±0.01	0.04±0.00	0.50±0.04	0.34±0.04	0.04±0.01	0.06±0.00
DiffTOP (Ours) + random init. .	0.18±0.02	0.68±0.03	0.67±0.01	0.19±0.04	0.04±0.00	0.39±0.04	0.30±0.02	0.00±0.00	0.05±0.01

Table 4: Comparison of all the methods on the Maniskill2 baseline. DiffTOP consistently outperforms both baselines on all tasks with action initialization from the BC policy. If zero or random initialization are used in DiffTOP, the performance drops, possibly due to the convergence to bad local minima during nonlinear trajectory optimization without a good action initialization.

In the original Diffusion Policy (Chi et al., 2023) paper, it was observed that the use of positional controllers yielded superior results for Diffusion Policy compared to the default velocity controller on Robomimic (Mandlekar et al., 2021) tasks. We evaluate Diffusion Policy, which is the strongest baseline, and DiffTOP on the most difficult three tasks with ph (proficient-human demonstration) and mh (multi-human demonstration) demonstrations using positional controller. The results with the positional controller are presented in Table 5. Diffusion Policy already achieves nearly the maximal possible performance on most tasks with the positional controller. DiffTOP, however, is able to achieve similar or even higher performances on most of these tasks.

	Square (ph)	Square (mh)	Transport (ph)	Transport (mh)	ToolHang (ph)
Diffusion	**0.98**±0.01	**0.97**±0.02	**1.00**±0.00	0.88±0.02	0.95±0.02
DiffTOP + Diffusion	**0.98**±0.01	0.96±0.02	**1.00**±0.00	**0.91**±0.01	**0.96**±0.01

Table 5: Performance Comparison of DiffTOP and Diffusion Policy using Positional Controllers on Robomimic Tasks.

Additionally, we do ablation experiments on the planning horizon H for imitation learning, with the results presented in Table 6. We observe that simply increasing the planning horizon H in imitation learning does not necessarily enhance performance. As the horizon increases from $H = 1$ to $H = 3$, the performance remains nearly the same; however, when H is increase to 5, we observe a slight decline in the performance.

B Implementation Details

In this section, we describe the implementation details of DiffTOP for the model-based RL experiments. For the imitation learning part, the code structure is very similar to this model-based RL implementation. For more detailed information, please refer to the code we will release upon acceptance of the paper. We implement DiffTOP on top of the open-source implementation of TD-MPC (Hansen et al., 2022) from the authors. Below we show the pseudo-code of the training function in DiffTOP.

	Square (ph)	Transport (ph)	ToolHang (ph)	Push-T
$H = 1$	**0.92**±0.01	**0.96**±0.01	**0.92**±0.01	**0.91**±0.01
$H = 3$	**0.92**±0.01	0.94±0.02	**0.92**±0.00	0.88±0.02
$H = 5$	0.91±0.01	0.94±0.01	0.90±0.00	0.88±0.01

Table 6: Ablation experiments for the planning horizon H in imitation learning.

```
def train():
    """
    Training code
    """
    for step in range(total_steps):
        obs = env.reset()
        # Differentiable trajectory optimization and update model
        action, info = agent.plan_theseus_update(obs)
        # Env step
        obs, reward, done, _ = env.step(action.cpu().numpy())
        # collect data in buffer and update model (TD-MPC loss)
        replay_buffer += (obs, action, reward, done)
        agent.update(replay_buffer)
```

Then, we demonstrate how the policy gradient loss is computed by differentiable trajectory optimization in DiffTOP with PyTorch-like pseudocode:

```
def plan_theseus_update(obs):
    """
    Differentiable trajectory optimization and update model using policy
    gradient loss.
    h, R, Q, d: model components.
    c0: loss coefficients.
    """
    import theseus as th

    # Encode first observation
    z = self.model.h(obs)

    # Get initialization action from pi
    init_actions = self.model.pi(z)

    # Theseus variable
    actions = th.Vector(tensor=actions, name="actions")
    obs = th.Variable(obs, name="obs")

    # Cost Function and Objective
    cost_function = th.AutoDiffCostFunction([obs], [action]
        ,value_cost_fn)
    objective = th.Objective().add(cost_function)

    # Trajectory optimization optimizer
    theseus_optim = th.TheseusLayer(th_optimizer)

    # Theseus layer forward
    theseus_inputs = {"actions": init_actions, "obs": obs}
    updated_inputs, info = theseus_optim.forward(theseus_inputs)
    updated_actions = updated_inputs['actions']

    # Update model using policy gradient losss
    a_loss = - torch.min(*self.model.Q_s(obs, updated_actions[0]))*c0
    a_loss.backward()
    optim_a.step()
```

For model-based reinforcement learning, We provide the network details for the added networks we used upon TD-MPC, which are the twin Q networks $\tilde{Q}_\phi$ learned in the original state space for computing the deterministic policy gradient.

```
(Q_s1): Sequential(
    (0): Linear(in_features=S, out_features=256)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=256, out_features=Z))
    (3): Linear(in_features=Z+A, out_features=512)
    (4): LayerNorm((512,), elementwise_affine=True)
    (5): Tanh()
    (6): Linear(in_features=512, out_features=512)
    (7): ELU(alpha=1.0)
    (8): Linear(in_features=512, out_features=1))
(Q_s2): Sequential(
    (0): Linear(in_features=S, out_features=256)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=256, out_features=Z))
    (3): Linear(in_features=Z+A, out_features=512)
    (4): LayerNorm((512,), elementwise_affine=True)
    (5): Tanh()
    (6): Linear(in_features=512, out_features=512)
    (7): ELU(alpha=1.0)
    (8): Linear(in_features=512, out_features=1))
```

For Imitation Learning, The default network details are as follows. Note that for Robomimic (Mandlekar et al., 2021) and Push-T tasks, we use the RNN-encoder from Robomimic; for ManiSkill (Mu et al., 2021; Gu et al., 2023) tasks, we use the PointNet encoder from ManiSkill2 Gu et al. (2023).

```
(ho): Sequential(
    (0): Linear(in_features=S, out_features=256)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=256, out_features=256)
    (3): ELU(alpha=1.0)
    (4): Linear(in_features=256, out_features=Zs))
(ha): Identity
(hl): Sequential(
    (0): Linear(in_features=Zs+A, out_features=256)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=256, out_features=256)
    (3): ELU(alpha=1.0)
    (4): Linear(in_features=256, out_features=128))
(R): Sequential(
    (0): Linear(in_features=Zs+A+64, out_features=512)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=512, out_features=512)
    (3): ELU(alpha=1.0)
    (4): Linear(in_features=512, out_features=1))
(d): Sequential(
    (0): Linear(in_features=Zs+A+64, out_features=512)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=512, out_features=512)
    (3): ELU(alpha=1.0)
    (4): Linear(in_features=512, out_features=Zs+64))
```

Hyperparameters used for DiffTOP for both model-based RL and imitation learning are shown in Tab 7. In model-based RL, we use the same parameters with TD-MPC (Hansen et al., 2022) whenever possible.

C Environment Details

For model-based reinforcement learning evaluation, we use 15 visual continuous control tasks from Deepmind Control Suite (DMC). For imitation learning, we use 13 tasks (detailed information can

Hyperparameter	Value
Model-based RL	
Max planning iterations	100 (50)
Planning step size	1e-4 (5e-3)
Discount factor	0.99
Action loss coefficient (c0)	1
optimizer	Adam($\beta_1 = 0.9$, $\beta_2 = 0.999$)
Gradient Norm	10
Planning horizon schedule	$1 \to 5$ (25k steps)
Batch size	256
Latent dimension	50
Sampling technique	PER($\alpha = 0.6$, $\beta = 0.4$)
Learning rate	1e-3
Imitation Learning	
Max planning iterations	100
Planning step size	1e-4
Planning horizon schedule	1
Latent dimension	50
Posterior Gaussian dimension	64
KL coefficien	1
Learning rate	3e-4
GMM Num Modes	5
RNN Seq Len	16
RNN Hidden Dim	1000
Point Cloud Sampled Points (ManiSkill)	1200

Table 7: Hyperparameters used in DiffTOP.

be found in Table 8) from Robomimic (Mandlekar et al., 2021), IBC (Florence et al., 2022), ManiSkillp (Mu et al., 2021), and ManiSkill2 (Gu et al., 2023).

Task	Source	Obs. Type	Ac Dim	Object	Demo	Task Description
Square	Robomimic	Img	7	Rigid	200	Pick a square nut and place it on a rod.
Transport	Robomimic	Img	14	Rigid	200	Transfer a hammer from a container to a bin
ToolHang	Robomimic	Img	7	Rigid	200	assemble a frame consisting of a base and hook
Push-T	IBC	Img	2	Rigid	200	Push a T-shaped object to a specified position
OpenCabinetDrawer	ManiSkill1	Point Cloud	13	Rigid	300/obj.	Open a specific drawer of the cabinet
OpenCabinetDoor	ManiSkill1	Point Cloud	13	Rigid	300/obj.	Open a specific door of the cabinet
PushChair	ManiSkill1	Point Cloud	22	Rigid	300/obj.	Push the swivel chair to the target position
MoveBucket	ManiSkill1	Point Cloud	22	Rigid	300/obj.	Move a bucket and lift it onto a platform
PickCube	ManiSkill2	Point Cloud	7	Rigid	1000	Pick up a cube and move it to a goal position
Fill	ManiSkill2	Point Cloud	7	Soft	200	Fill clay from a bucket into the target beaker
Hang	ManiSkill2	Point Cloud	7	Soft	200	Hang a noodle on a target rod
Excavate	ManiSkill2	Point Cloud	7	Soft	200	Lift a amount of clay to a target height
Pour	ManiSkill2	Point Cloud	7	Soft	200	Pour liquid from a bottle into a beaker

Table 8: Imitation Learning Tasks Summary.

We visualize the keyframes of the imitation learning tasks in Fig 8.

D MORE IMPLEMENTATION DETAILS ON USING CVAE FOR IMITATION LEARNING

We provide more details on how we instantiate DiffTOP with CVAE in imitation learning, in which the goal is to reconstruct the expert actions conditioned on the state. The CVAE encoder is composed of three networks: the first network is a state encoder h_θ^o that encodes the state into a latent feature vector $z^s = h_\theta^o(s_i)$, which is the conditional information in our case. The second is an action encoder h_θ^a that encodes the expert action into a latent feature vector $z^a = h_\theta^a(a_i^*)$. The last is a fusing encoder $h_\theta^l(z^s, z^a)$ that takes as input the concatenation of the state and action latent features, and outputs the mean μ and variance σ^2 of the posterior Gaussian distribution $\mathcal{N}(\cdot|\mu, \sigma^2)$. During training, the final latent state z for state s_i used in Equation 7 is the concatenation of a sampled vector $\tilde{z}$ from the posterior Gaussian distribution $\mathcal{N}(\cdot|\mu, \sigma^2)$, and the latent state feature vector z^s: $z = [\tilde{z}, z^s], \tilde{z} \sim \mathcal{N}(\cdot|\mu, \sigma^2)$.

The latent state z will then be used as input for the decoder, which consists of the reward function R_θ, and the dynamics function d_θ. Trajectory optimization is performed with the reward and dynamics function in the decoder to solve Equation 7 to generate the reconstructed action $a^*(\theta; s_i)$. The loss for training the CVAE is the evidence lower bound (ELBO) on the demonstration data:

$$\mathcal{L}_{DiffTOP}^{IL}(\theta) = \sum_{i=1}^{N} ||a(\theta; s_i) - a_i^*||_2^2 - \beta \cdot \text{KL}(\mathcal{N}(\cdot|\mu, \sigma^2)|\mathcal{N}(0, I)), \tag{9}$$

where $\text{KL}(P||Q)$ denotes the KL divergence between distributions P and Q. At test time, only the decoder of the CVAE is used for generating the actions. Given a state s, the latent state z is the concatenation of the encoded latent state feature z^s, and a sampled vector $\tilde{z}$ from the prior distribution $\mathcal{N}(0, 1)$.

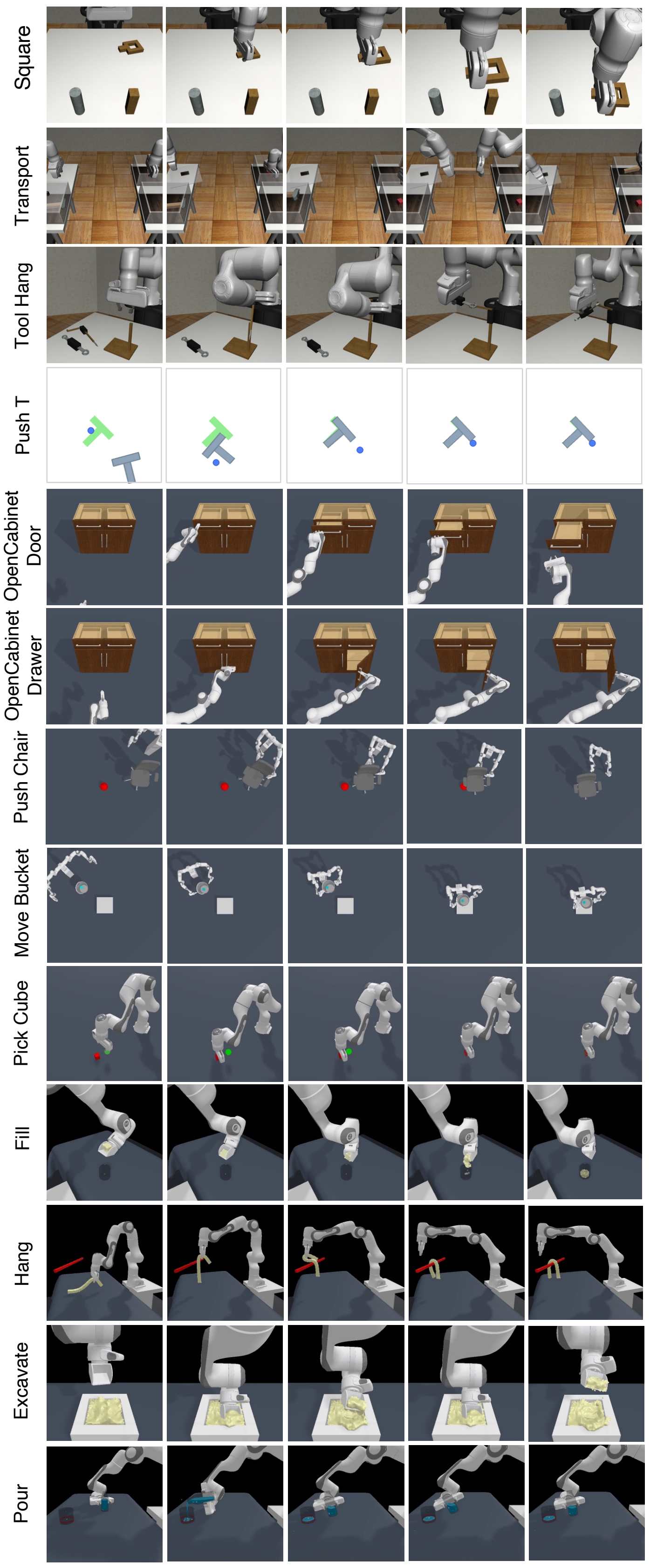

Figure 8: Visualization of the tasks for imitation learning.