A Additional Results

In addition to the aggregated results in the main paper, we also provide per-task results for all experiments and tasks in simulation. Our benchmark results are shown in Figure 7, and task transfer results are shown in Figure 10. Per-task results for all of our ablations are shown in Figure 8 and Figure 9.

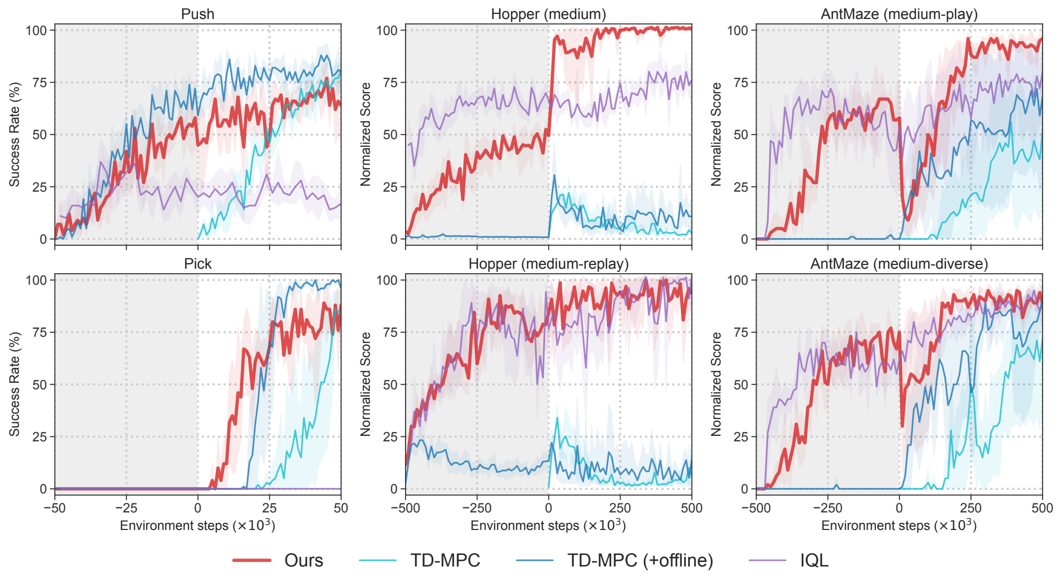


Figure 7. **Comparison of our method against baselines.** Offline pretraining is shaded gray. Mean of 5 seeds; shaded area indicates 95% CIs.

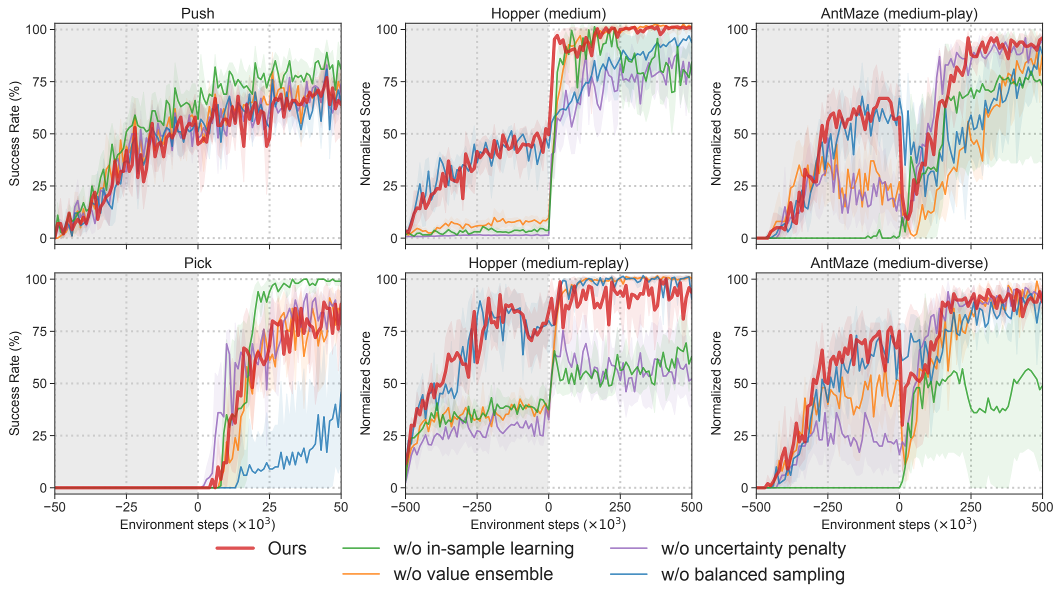


Figure 8. **Ablation results on all tasks.** Offline pretraining is shaded gray. Mean of 5 seeds; shaded area indicates 95% CIs.

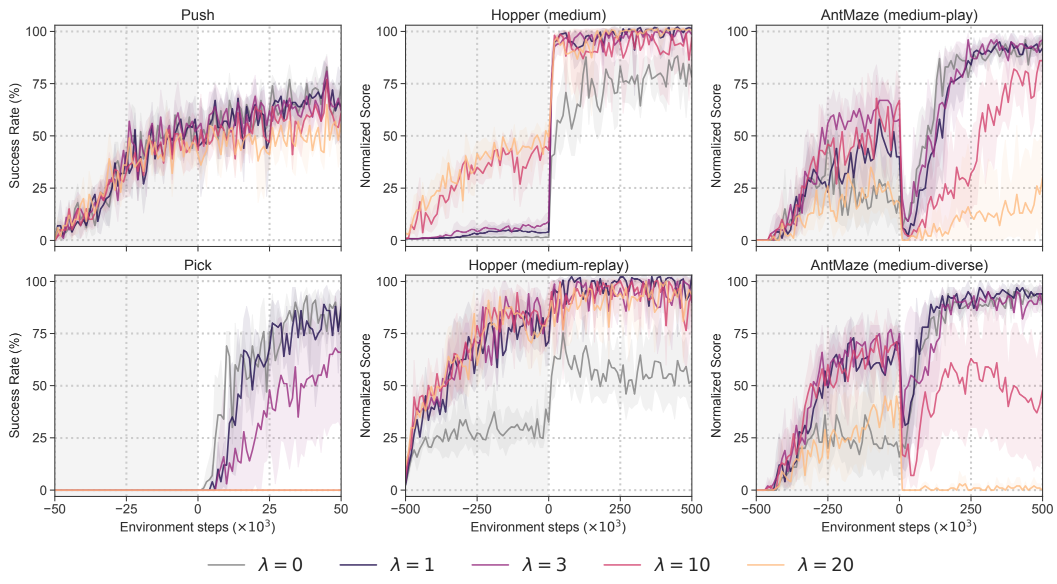


Figure 9. **Ablation study on uncertainty coefficient (λ).** Offline pretraining is shaded gray. Mean of 5 seeds; shaded area indicates 95% CIs.

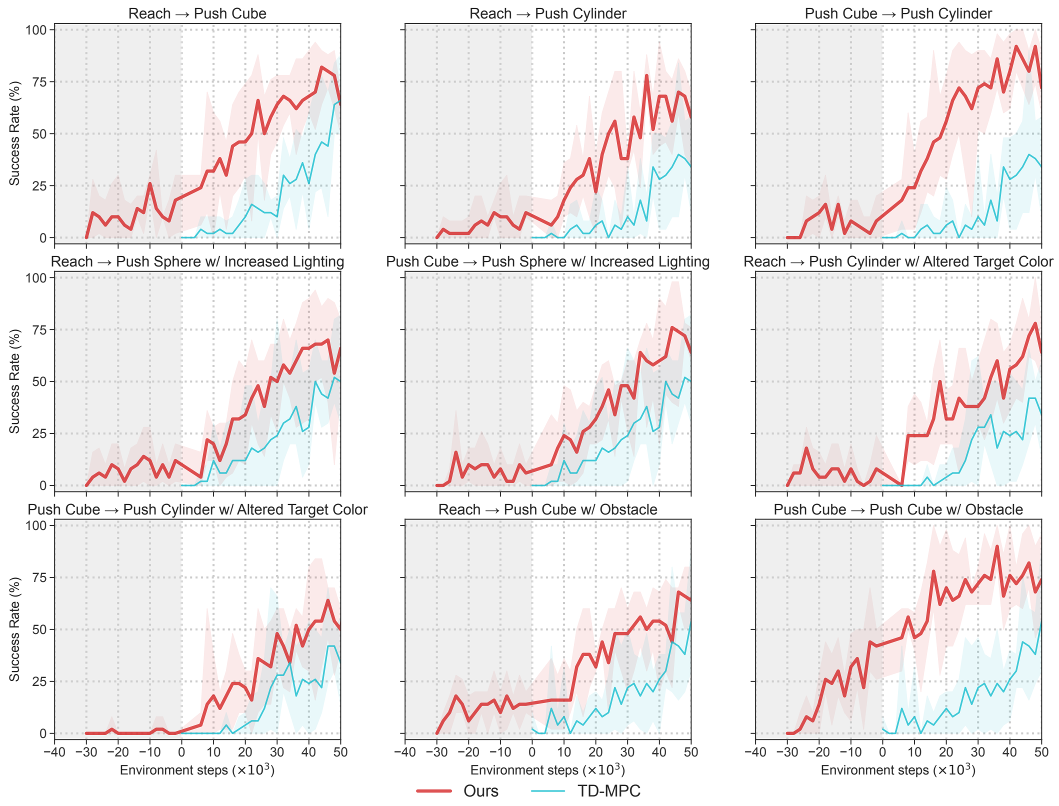


Figure 10. **Task transfer results.** Success rate (%) of our method and TD-MPC trained from scratch on all 9 simulated transfer tasks. We designed these tasks based on the xArm [27] task suite. Offline pretraining is shaded gray. Mean of 5 seeds; shaded area indicates 95% CIs.

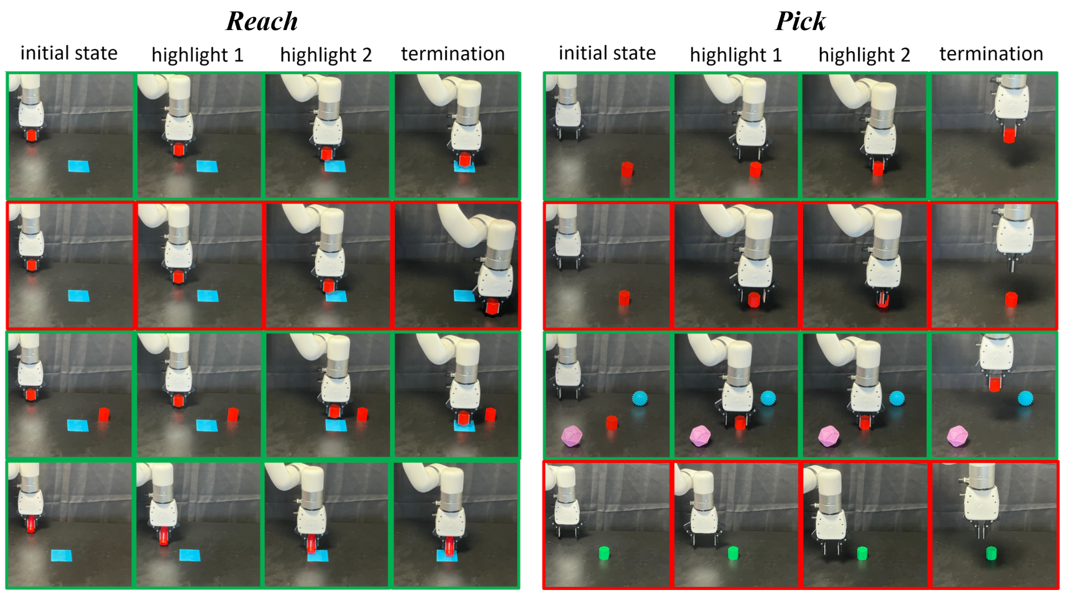


Figure 11. **Sample trajectories.** We include eight trajectories from the offline dataset or evaluation results, which illustrate all real-world tasks considered in this work. Successful trajectories are marked green while failed trajectories are marked red.

B Tasks and Datasets

B.1 Real-World Tasks and Datasets

We implement two visuo-motor control tasks, *reach* and *pick* on a UFactory xArm 7 robot arm with an Intel RealSense Depth Camera D435 as the only external sensor. The observation space contains a 224 × 224 RGB image and an 8-dimensional robot proprioceptive state including the position, rotation, and the opening of the end-effector and a boolean value indicating whether the gripper is stuck. Both tasks are illustrated in Figure 3 *(second from the left)*. For safety reasons, we limit the moving range of the gripper in a $30cm \times 30cm \times 30cm$ cube, of which projection on the table is illustrated in Figure 12. To promote consistency between experiments, we evaluate agents on a set of fixed positions, visualized as red crosses in the aforementioned figure.

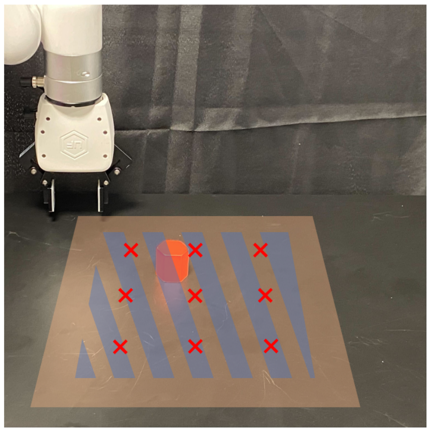


Figure 12. **Real-world workspace.** Moving range of the end-effector and the initialization range of target/object are shaded in the image. The positions for evaluation are labeled by crosses.

Reach The objective of this task is to accurately position the red hexagonal prism, held by the gripper, above the blue square target. The action space of this task is defined by the first two dimensions, which correspond to the horizontal plane. The agent will receive a reward of 1 when the object is successfully placed above the target, and a reward of 0 otherwise. The offline dataset for *reach* comprises 100 trajectories collected using a behavior-cloning policy, which exhibits an approximate success rate of 50%. Additionally, there are 20 trajectories collected through teleoperation, where the agent moves randomly, including attempts to cross the boundaries of the allowable end-effector movement. These 20 trajectories are considered to be diverse and are utilized for conducting an ablation study around the quality of the offline dataset.

Pick The objective of this task is to grasp and lift a red hexagonal prism by the gripper. The action space of this task contains the position of the end-effector and the opening of the gripper. The agent will receive a reward of 1 when the object is successfully lifted above a height threshold, a reward of 0.5 when the object is grasped but not lifted, and a reward of 0 otherwise. The offline dataset for *pick* comprises 200 trajectories collected using a BC policy that has an approximate success rate of 50%. Figure 11 shows sample trajectories from our offline dataset for *pick*.

Real-world transfer tasks We designed two transfer tasks for both *reach* and *pick*, as shown in Figure 3 *(the second from right)*. As the red hexagonal prism is an important indicator of the end-effector position in *reach*, we modify the task by (1) placing an additional red hexagonal prism on the table, alongside the existing one, and (2) replacing the object with a small red ketchup bottle, whose bottom is not aligned with the end-effector. In *pick*, the red hexagonal prism is regarded as a target object. Therefore we (1) add two distractors, each with a distinct shape and color compared to the target object, and (2) change the color and shape of the object (from a red hexagonal prism to a green octagonal prism). We've shown by experiments that different modifications will have different effects on subsequent performance in finetuning, which demonstrates both the effectiveness and limitation of the offline-to-online pipeline we discussed.

B.2 Simulation Tasks and Datasets

xArm *Push* and *pick* are two visuo-motor control tasks in the xArm robot simulation environment [27] implemented in MuJoCo. The observations consist of an 84×84 RGB image and a 4-dimensional robot proprioceptive state including the position of the end-effector and the opening of the gripper. The action space is the control signal for this 4-dimensional robot state. The tasks are visualized in Figure 3 *(left)*. *push* requires the robot to push a green cube to the red target. The goal in *pick* is to pick up a cube and lift it above a height threshold. Handcrafted dense rewards are used for these two tasks. We collected the offline data for offline-to-online finetuning experiments by training TD-MPC agents from scratch on these tasks. We saved the first 40k environment steps (800 trajectories) in the replay buffer as an offline dataset for each task. Figure 13 gives an overview of the offline data distribution for the two tasks.

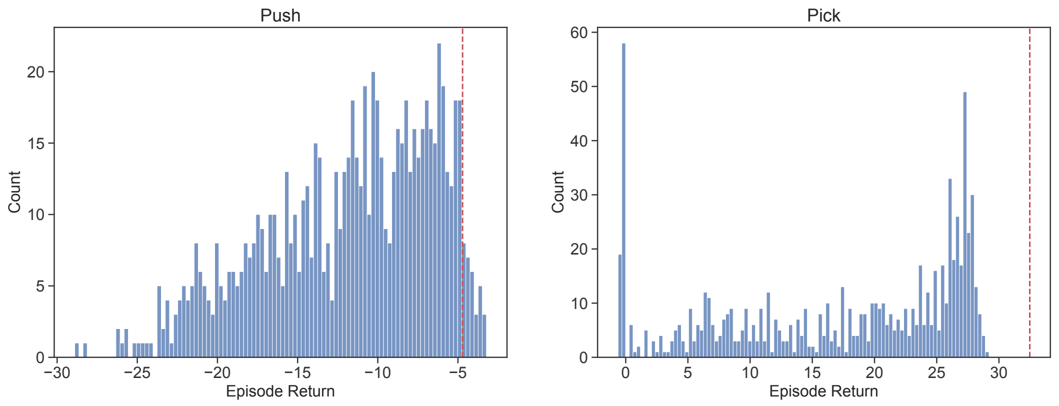
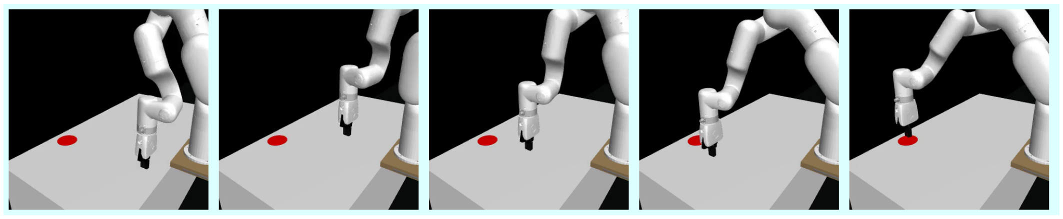
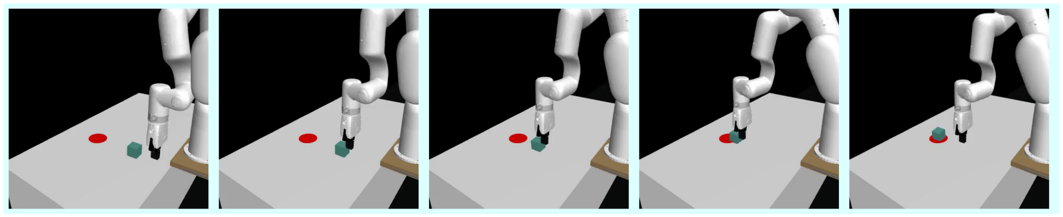


Figure 13. **Offline dataset statistics for xArm tasks in simulation.** We plot the distribution of episode returns for trajectories in the two offline datasets. The red line indicates the mean performance achieved by our method after online finetuning.

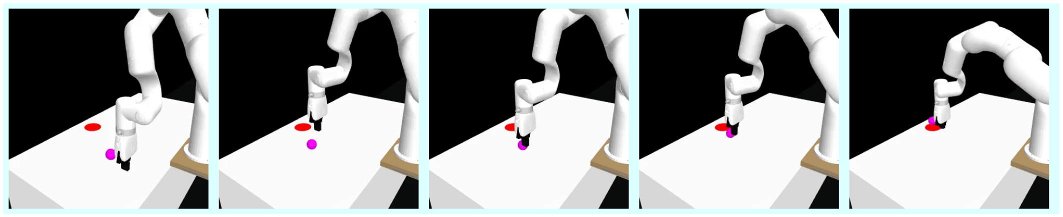
Transfer tasks We designed nine transfer tasks based on *reach* (the same task as real *reach* but simplified because of the knowledge of ground-truth positions) and *push* in simulation to evaluate the generalization capability of offline pretrained model. Compared to real-world tasks, the online budget is abundant in simulation, thus we increase the disparity between offline and online tasks such as finetuning on a totally different task. As the target point for both tasks is a red circle, we directly use *reach* as offline pretrain task and online finetuning on different instances of *push* including push cube, push sphere, push cylinder, and push cube with an obstacle. All these tasks are illustrated in Figure 14.



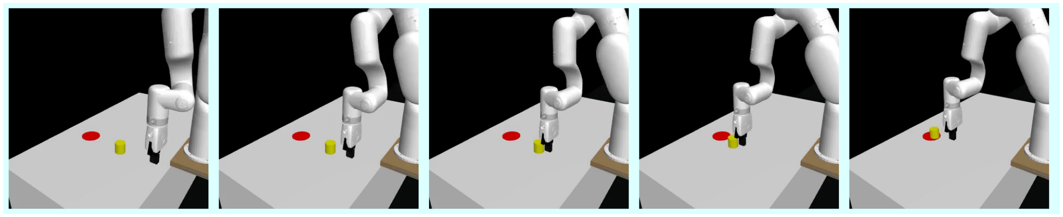
(a) Reach.



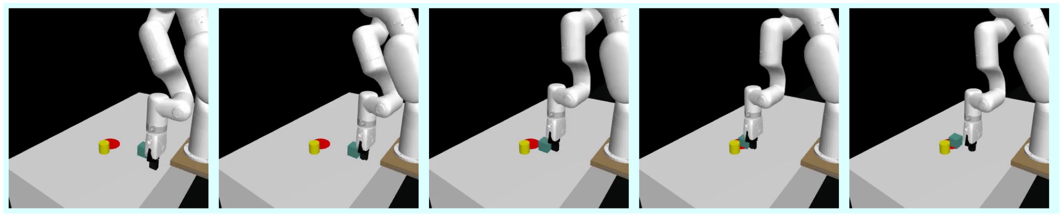
(b) Push.



(c) Push sphere with increased lighting.



(d) Push cylinder.



(e) Push with obstacle.

Figure 14. **Transfer tasks in our simulated xArm environments.** We consider a total of 9 transfer settings in simulation. We here visualize a trajectory for each of the tasks used in our xArm experiments. Task labels correspond to those shown in Figure 10.

D4RL We consider four representative tasks from two domains (Hopper and AntMaze) in the D4RL [26] benchmark. Each domain contains two data compositions. *Hopper* is a Gym locomotion domain where the goal is to make hops that move in the forward (right) direction. Observations contain the positions and velocities of different body parts of the hopper. The action space is a 3-dimension space controlling the torques applied on the three joints of the hopper. *Hopper (medium)* uses 1M samples from a policy trained to approximately 1/3 the performance of the expert, while *Hopper (medium-replay)* uses the replay buffer of a policy trained up to the performance of the medium agent. *Antmaze* is a navigation domain with a complex 8-DoF quadruped robot. We use the *medium* maze layout, which is shown in Figure 3 *(left)*. The *play* dataset contains 1M samples generated by commanding specific hand-picked goal locations from hand-picked initial positions, and the *diverse* dataset contains 1M samples generated by commanding random goal locations in the maze and navigating the ant to them. This domain is notoriously challenging because of the need to "stitch" suboptimal trajectories. These four tasks are officially named `hopper-medium-v2`, `hopper-medium-replay-v2`, `antmaze-medium-play-v2` and `antmaze-medium-diverse-v2` in the D4RL benchmark.

C Implementation Details

Q-ensemble and uncertainty estimation We provide PyTorch-style pseudo-code for the implementation of the Q-ensemble and uncertainty estimation discussed in Section 3.2. Here `Qs` is a list of Q-networks. We use the minimum value of two randomly selected Q-networks for Q-value estimation, and the uncertainty is estimated by the standard deviation of all Q-values. We use five Q-networks in our implementation.

```
def Q_estimate(Qs, z, a):
    x = torch.cat([z, a], dim=-1) # concatenate (latent) state and action
    idxs = random_choice(len(Qs), 2, replace=False) # randomly select two distinct Qs
    q1, q2 = Qs[idxs[0]](x), Qs[idxs[1]](x)
    return torch.min(q1, q2)  # return the minimum of the two as Q value estimation

def Q_uncertainty(Qs, z, a):
    x = torch.cat([z, a], dim=-1) # concatenate (latent) state and action
    qs = torch.stack(list(q(x) for q in Qs), dim=0)
    uncertainty = qs.std(dim=0)   # compute the standard deviation as uncertainty
    return uncertainty
```

Network architecture For the real robot tasks and simulated xArm tasks where observations contain both an RGB image and a robot proprioceptive state, we separately embed them into feature vectors of the same dimensions with a convolutional neural network and a 2-layer MLP respectively, and do element-wise addition to get a fused feature vector. For D4RL tasks where observations are state features, only the state encoder is used. We use five Q-networks to implement the Q-ensemble for uncertainty estimation. All Q-networks have the same architecture. An additional V network is used for state value estimation as discussed in Section 3.1.

Hyperparameters We list the hyperparameters of our algorithm in Table 5. The hyperparameters related to our key contributions are highlighted.

Other details We apply image shift augmentation [49] to image observations, and use Prioritized Experience Replay (PER; [50]) when sampling from replay buffers.

D Baselines

TD-MPC We use the same architecture and hyperparameters for our method and our two TD-MPC baselines as in the public TD-MPC implementation from `https://github.com/nicklashansen/tdmpc`, except that we use two encoders, one for each modality, in the real robot and xArm tasks that use both visual inputs and robot proprioceptive information. For the **TD-MPC (+offline)** baseline, we naïvely pretrain the model on offline data and then finetune it with online RL without any changes to hyperparameters.

IQL We use the official implementation from `https://github.com/ikostrikov/implicit_q_learning` for the IQL baseline. We use the same hyperparameters that the authors used for D4RL tasks. For xArm tasks, we perform a grid search over the hyperparameters $\tau \in$

Table 5. **Hyperparameters.**

Hyperparameter	Value
Expectile (τ)	0.9 (AntMaze, xArm)
	0.7 (Hopper)
AWR temperature (β)	10.0 (AntMaze)
	3.0 (Hopper, xArm)
Uncertainty coefficient (λ)	1 (xArm)
	3 (AntMaze)
	20 (Hopper)
Q ensemble size	5
Batch size	256
Learning rate	3e-4
Optimizer	Adam($\beta_1 = 0.9, \beta_2 = 0.999$)
Discount	0.99 (D4RL)
	0.9 (xArm)
Action repeat	1 (D4RL)
	2 (xArm)
Value loss coefficient	0.1
Reward loss coefficient	0.5
Latent dynamics loss coefficient	20
Temporal coefficient	0.5
Target network update frequency	2
Polyak	0.99
MLP hidden size	512
Latent state dimension	50
Population size	512
Elite fraction	50
Policy fraction	0.1
Planning iterations	6 (xArm)
	1 (D4RL)
Planning horizon	5
Planning temperature	0.5
Planning momentum coefficient	0.1

$\{0.5, 0.6, 0.7, 0.8, 0.9, 0.95\}$ and $\beta \in \{0.5, 1.0, 3.0, 10.0\}$, and we find that expectile $\tau = 0.95$ and temperature $\beta = 10.0$ achieves the best results. We add the same image encoder as ours to the IQL implementation in visuo-motor control tasks.