APPENDIX

A ADDITIONAL EXPERIMENTS

Data scaling. In this section, we present additional scaling experiments for the scaling portion of Section 4.1 and the temporally correlated noise portion of Section 5. We present the performance of the methods for each task instead of an average over the tasks.

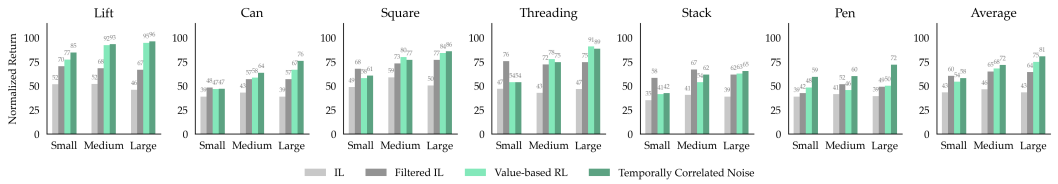

Figure 11: **Normalized returns** of value-based RL compared with IL, filtered-IL, and temporally-correlated noise at different data scales, shown for each task.

From Figure 11, we see that value-based RL scales better with data in nearly every task, while IL-based methods either do not scale or scale in a limited capacity. In addition, temporally-correlated noise outperforms not adding temporally correlated noise for each task and data scale. Temporally correlated noise is especially useful for Adroit `Pen`, which has been known in the literature to benefit from more exploration.

Value-based RL with more iterations for Square and Stack. Because of compute restrictions, the results reported in the main paper for Robomimic `Square` and MimicGen `Stack` were converged for IL-based methods but not for value-based RL. We report the results for value-based RL run for a longer number of iterations in Figure 12. We see that the difference between IL methods and value-based RL becomes larger with more iterations.

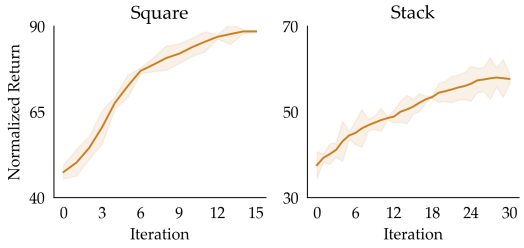

Figure 12: **Normalized returns of value-based RL** for Robomimic `Square` and MimicGen `Stack`. Error bars show standard error over 3 seeds.

Value-based RL with more rollouts per iteration. We also report runs for value-based RL with more rollouts per iterations for environments that saturated prematurely. From Figure 13, we see that value-based RL often exceeds premature saturation just with more data in each iteration.

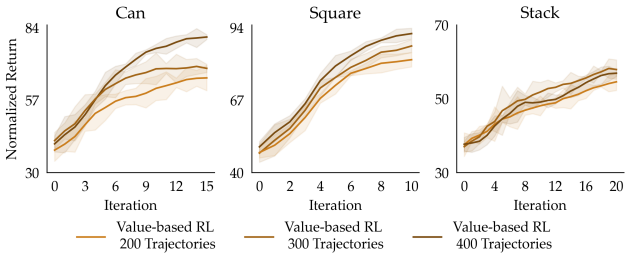

Figure 13: **Normalized returns of value-based RL** for more rollouts per iteration for Robomimic `Can`, `Square` and MimicGen `Stack`. Error bars show standard error over 3 seeds.

B EXPERIMENT DETAILS

Training Parameters. We set the number of iterations from N=10 to 20 depending on the environment and M=200 rollouts per iteration. We choose the number of trajectories in the initial dataset such that the base IL policy can get 30 to 65% normalized returns prior to batch online RL. For temporally correlated Ornstein-Uhlenbeck (OU) noise, we select one θ and σ value for each environment suite. For implementation, we use the same residual block structure for the policy as IDQL (Hansen-Estruch et al., 2023) for both expressive policy RL and imitation learning. We use a simple MLP for Gaussian policies.

Tasks	Parameters	Values
Robomimic `Lift`	Dataset Size	5
	OU θ	5
	OU σ	0.05
Robomimic `Can`	Dataset Size	10
	OU θ	5
	OU σ	0.05
Robomimic `Square`	Dataset Size	100
	OU θ	5
	OU σ	0.05
MimicGen `Stack`	Dataset Size	20
	OU θ	5
	OU σ	0.03
MimicGen `Threading`	Dataset Size	50
	OU θ	5
	OU σ	0.03
Adroit `Pen`	Dataset Size	3
	OU θ	0.1
	OU σ	0.03
All Tasks	Batch Size	256
	Learning Rate	3e-4
	IQL Expectile	0.8
	Discount	0.99
	Number of Sampled Actions	64
	Optimizer	Adam
	Beta Schedule	Variance Preserving
	Diffusion Steps	100
	Diffusion Policy: MLP Hidden Dim	256
	Diffusion Policy: Num Residual Blocks	3
	Gaussian Policy: MLP Hidden Dim	256
	Gaussian Policy: MLP Hidden Layers	3

Table 1: Hyperparameters for each simulation task. The values specified under All Tasks are shared for different tasks.

Data Sources. For each task, the dataset consists of expert trajectories. In Robomimic tasks, we use the Proficient Human dataset provided by Mandlekar et al. (2021). In MimicGen environments, we use the dataset provided by the benchmark (Mandlekar et al., 2023). For Adroit, we use the dataset from D4RL (Fu et al., 2020).

Evaluation Protocol. Evaluations are performed by rolling out the policy from start states randomly sampled from the default initial state distribution of the task. The rollout length for `Lift`, `Can`, and `Square` is 400; for `Stack` is 200; for `Threading` is 400; and for `Pen` is 100. Results in the main text report normalized return averaged over 3 seeds and 100 evaluation trials each.

C Real-World Task Details

In this section, we provide more information on the real world Tape task in our analysis.

Setup Description. The Tape task involves hanging a roll of tape onto a rack by controlling a 7-DoF Franka Research 3 robot. To successfully complete the task, the robot needs to precisely aim for and grasp the roll of tape and hang it to the hook. The initial distribution is a roughly 15 cm $\times$ 15 cm area. We illustrate an example initial state, success state, and the initial state distribution in Figure 14. The RL agent sends actions to the robot at 5Hz with a maximum episode length of 200 timesteps. The robot obtains visual RGB input from two Intel RealSense D435 cameras, one on the mounted on the end effector and one mounted on the side.

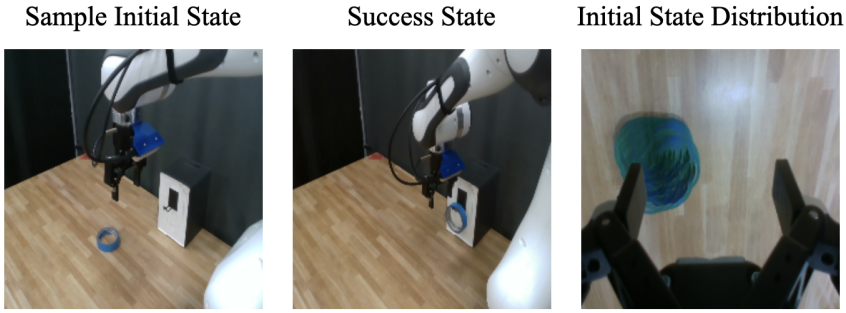

Figure 14: Scenes showing sample initial and success state and the initial state distribution of the real-world Tape task.

Success Detection. The Tape task contains a success state that must be reached for the rollout to be considered successful, namely having the tape on the rack. We use a scripted rule to detect if this state has been reached and if there is a success. For each environment step, we utilize a color threshold to check the color of the pixel above the hook. We manually select the pixel location and verify the error of the success detection is near zero.

Resets. We perform automatic resets of the Tape environment in our experiments. For a successful rollout, we replay a pre-recorded trajectory to grasp the tape and lift it off the hook. For a failed rollout, we detect the location of the tape and execute a primitive to lift the tape. In both cases, after lifting the tape, we sample an initial state from the initial state distribution and place the tape at the initial state location for the next episode.