

A Additional Experiments

A.1 Model-Based Generalization

The "partial" task also provides a good test bed for an algorithm's generalization capabilities, since the offline dataset does not contain full solutions for it. This is a different problem than the standard dynamic programming ("stitching") issue of data-centric reinforcement learning since the dataset does not contain a sequence of state-action pairs that lead from the initial state to the goal state. Instead, to solve this task, a learning agent must understand the compositional nature of the scene and do combinatorial generalization over the objects. In this section we seek to answer whether 1) the learned model can do combinatorial generalization of within distribution tasks and 2) whether policy optimization can take advantage of the model's capabilities. We evaluate the agent at the end of the offline pre-training phase. To answer the first question, we consider episodes that successfully complete the "partial" task from the trained agent. We condition our model on the frames that solves the first three tasks (which are covered in the offline dataset) and rollout the expert actions to predict the following frames. Results are shown in Fig. 1. The model successfully predicts a combination of the microwave, kettle, bottom burner and light switch in the correct configuration, despite never encountering these four objects together in the offline dataset. Moreover, we evaluate the model-predicted rewards on these expert trajectories, plotted in Fig. 5 (left). We see that the model predicts rewards of up to 4 with an average reward of 3.63, despite only being trained on trajectories with maximum reward of 3. This results show that the learned model is capable of compositional generalization. To evaluate whether the learned policy can take advantage of the model generalization capabilities, we rollout the trained agent under the model and evaluate the predicted rewards, results are shown in Fig. 5 (right). The agent achieves an average final reward of 3.52 under the learned model and solves all four tasks. This suggest that the model-based RL agent is able to do combinatorial generalization, but the offline dataset is not enough to adequately learn the environment dynamics.

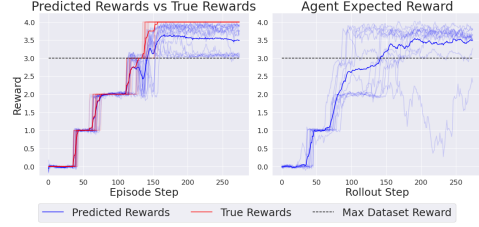


Figure 5: We evaluate the model's generalization capabilities at the end of the offline pre-training phase. The model correctly predicts rewards of up to 4 on successful episodes in the "partial" task, even though the maximum dataset reward is 3. (left). When doing rollouts in the learned model, the policy solves all four objects in the "partial" task and reaches rewards of up to 4 (right).

B Theoretical Results and Empirical Validation

Theoretical Results for Uncertainty-Aware Model-based Training Given our choice of variational parametrization and model uncertainty estimation we can directly adapt certain theoretical guarantees from prior model-based RL literature [11, 20, 19]. We consider the following result in particular: let $T_\theta(s'|s, \mathbf{a})$ and $T(s'|s, \mathbf{a})$ be the learned and true latent dynamics models respectively. We define the discounted state-action distribution

$$\rho_{\mathcal{T}, \mu_0}^\pi(\mathbf{s}, \mathbf{a}) \propto \sum_{t=0}^{\infty} \gamma^t \mathbb{P}_{\mathcal{T}, \mu_0}^\pi(\mathbf{s}_t = \mathbf{s}) \pi(\mathbf{a}|\mathbf{s})$$

in the standard way. The function $u(\mathbf{s}, \mathbf{a})$ is an admissible error estimator if

$$d_{\mathcal{F}}[T(s'|s, \mathbf{a}) || T_\theta(s'|s, \mathbf{a})] \leq u(\mathbf{s}, \mathbf{a}).$$

For any policy π we can then define

$$\epsilon_u(\pi) = \mathbb{E}_{(\mathbf{s}, \mathbf{a}) \sim \rho_{\mathcal{T}, \mu_0}^\pi} [u(\mathbf{s}, \mathbf{a})].$$

The following Theorem then holds:

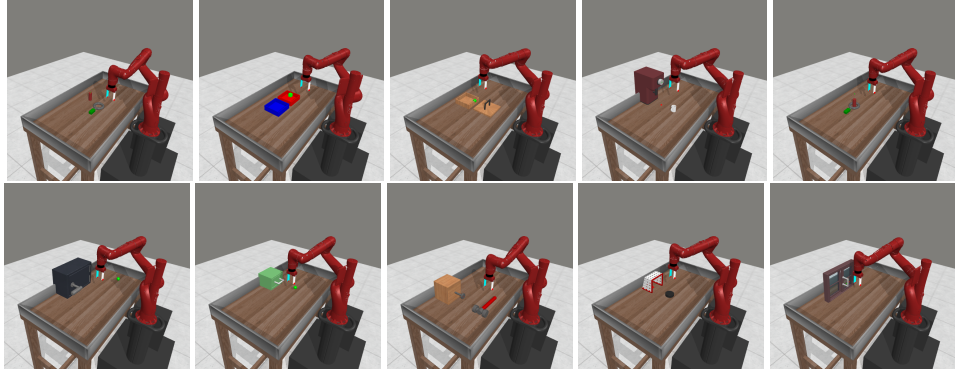


Figure 7: Visualization of the 10 different MetaWorld environments used in our experiments. Top row from left to right: assembly-v2, bin-picking-v2, box-close-v2, coffee-push-v2, disassemble-v2. Bottom row from left to right: door-open-v2, drawer-open-v2, hammer-v2, plate-slide-v2, window-open-v2.

Theorem B.1. (Informal) Let $\hat{\pi}^*(s)$ be the optimal policy under the learned model $T_\theta(s'|s, \mathbf{a})$ with an uncertainty-penalized reward and π^* the optimal policy in the ground-truth MDP. Under certain mild assumptions, then the following inequality holds:

$$2\alpha\epsilon_u(\pi^*) \geq \mathbb{E}_{\pi^*, T} \left[\sum_{t=0}^{\infty} r_t \right] - \mathbb{E}_{\hat{\pi}^*, T} \left[\sum_{t=0}^{\infty} r_t \right] \quad (11)$$

Proof. Consult [11]. □

Empirical verification From the Theorem, we can deduce that the policy under-performance is upper bounded by the discounted model-based uncertainty over the state-action distribution induced by the expert policy under the learned model. In practice we do not have access to an oracle estimator $u(s, \mathbf{a})$ and we use the ensemble disagreement from Eq. 2. While these results are not new, empirical verification is difficult in the fully offline case, since we have a static dataset, and all values are point estimates. However, in the online fine-tuning case, we have a continuum of datasets and we can empirically verify the claims of Theorem B.1.

We periodically evaluate $\epsilon_u(\pi^*)$ and the expected model uncertainty induced under the expert state-action distribution in the learned model. At each epoch E , we cannot generate model rollouts from the expert, since that would require training an expert policy under the current inference model q_{θ_E} . However, we can sample expert episodes from the trained expert and the environment. Given an expert trajectory $\tau^{\text{exp}} = \mathbf{x}_{1:T}, \mathbf{a}_{1:T}$ we sample latent belief states from the first $T - H$ steps to obtain $\mathbf{s}_{1:(T-h)} \sim q_{\theta_E}(\cdot | \mathbf{x}_{1:T-H}, \mathbf{a}_{1:T-H})$. From each state \mathbf{s}_j we then rollout the expert actions $\mathbf{a}_{j:j+H}$ using the current iteration of the dynamics model T_{θ_E} and obtain states $\{(\hat{\mathbf{s}}_j^t, \mathbf{a}_j^t)\}_{j=1, t=0}^{T-H, H}$ as in Section 4 (here $\mathbf{a}_j^t = \mathbf{a}_{j+t}$ from the expert dataset). We can then obtain the empirical estimate of

$$\epsilon_u(\pi^*) \approx \mathbb{E}_{q_{\theta_E}(\mathbf{s}_j^0 | \tau^{\text{exp}}), T_{\theta_E}} \left[\frac{1}{H(T-H)} \sum u_\theta(\hat{\mathbf{s}}_j^t, \mathbf{a}_j^t) \right] \quad (12)$$

Empirical results evaluated on the "partial" task are shown in Fig. 6. We see that the performance gap is strongly bounded (up to a choice of the penalty scale) by the estimate from Eq. 12, which verifies the claim of Theorem B.1.

C Experimental Details

C.1 Environments

The Franka Kitchen environment from [23] (RPL) is a challenging long-range control problem, which involves a simulated 9-DOF Franka Emika Robot in a kitchen setting. The robot uses joint-space control and the observation is a single 64x64 RGB image; we do not assume access to object states or robot proprioception. The goal of the agent is to manipulate a set of 4 pre-defined objects and receives a reward of 1.0 for each object in right configuration at each time step. This is a very challenging environment due to 1) high-dimensional observation spaces; 2) partial observability with non-trivial object and robot state estimation; 3) need for very-fine-grained control in order to operate the small elements of the environment, such as turning knobs and flipping the light-switch; 4) the long-range nature of the tasks; 5) the use of sparse rewards, which provide limited intermediate supervision to the policy, and finally 6) the use of high-dimensional control which requires learning forward kinematics from images alone. For our experiments we render the original RPL datasets and consider two environments from the D4RL benchmark [24]. The "mixed" task requires operating the microwave, kettle, light switch and slide cabinet and has a small set of successful demos in the offline dataset. The "partial" task, which requires manipulating the microwave, kettle, bottom burner and light switch does not have any trajectories that successfully complete all four objects, but has demonstrations for several configurations which complete up to three objects. We will release this dataset with our project to facilitate the development and testing of vision-based offline RL algorithms.

For the model-free methods, since we use a feedforward network for encoding images, we use a framestack of 3 for all of our model-free experiments. At each timestep t , the agent was provided with a history of the previous 3 images (from the offline trajectories during offline training, or from the environment during online training). For COMBO and LOMPO, since the latent dynamics model has a recurrent component and therefore can implicitly retain a history of observations, we did not use any framestacking with the image observations from the environments.

On the Franka Kitchen Environment, we did not use an action repeat, and on the Metaworld environments and data we used an action repeat of 2. For the online finetuning experiments, we used the following procedure: roll out the current policy in the environment for a single episode, add that episode to the replay buffer, and then finetuning the model, critic network, and the policy network. On the Franka Kitchen environment, after each episode we performed 50 gradient steps on each component of each method (eg: model, critic network, and the policy network). For the Metaworld environments, we performed 20 gradient steps after each episode. In total, on the Franka Kitchen environments, we performed 10,000 gradient steps of offline training and 66,300 gradient steps of online finetuning. On the Metaworld environments, we performed 1,000 gradient steps of offline training and 20,000 gradient steps of online finetuning.

C.2 Datasets

Kitchen

- Number of trajectories: 563
- Number of transitions: 128,569
- Average undiscounted episode return: 261.12
- Average number of objects manipulated per episode: 3.98

MetaWorld All of the MetaWorld datasets have 9 – 10 trajectories and 1,010 total transitions. The average undiscounted episode returns and success rates are shown in Table 1:

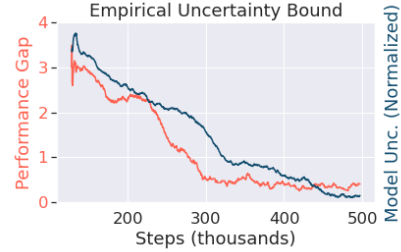


Figure 6: Empirical evaluation of Theorem B.1. We plot the performance gap versus the the empirical estimates of (normalized) expected model uncertainty using Eq. 12.

Environment	Avg. Return	Success Rate
assembly-v2	36.000	1.000
bin-picking-v2	20.900	1.000
box-close-v2	25.300	1.000
coffee-push-v2	36.200	1.000
disassemble-v2	31.556	1.000
door-open-v2	15.200	1.000
drawer-open-v2	48.000	1.000
hammer-v2	63.333	1.000
plate-slide-v2	71.100	1.000
window-open-v2	60.500	1.000

Table 1: Undiscounted episode returns and success rates in the MetaWorld datasets.

524 C.3 Model Based Methods

525 MOTO uses the model architecture from [32]. For the convolutional image encoder network, we use
526 the following hyperparameters:

- 527 • channels: (48, 96, 192, 384)
- 528 • kernel sizes: (4, 4, 4, 4)
- 529 • strides: (2, 2, 2, 2)
- 530 • padding: VALID
- 531 • four final MLP layers of size: 400

532 The decoder network consists of Deconvolution/Transpose convolution layers with the following
533 hyperparameters:

- 534 • four initial MLP layers of size: 400
- 535 • channels: (128, 64, 32, 3)
- 536 • kernel sizes: (5, 5, 6, 6)
- 537 • strides: (2, 2, 2, 2)
- 538 • padding: VALID

539 MOTO was trained using a model learning rate of 1×10^{-4} . The critic and policy network learning
540 rates are 8×10^{-5} . The batch size for model training is 16 and the batch size for agent training is
541 128. We also used a filtered behavioral cloning factor of 10 and a disagreement penalty factor of 10.

542 The latent dynamics model is represented using an RSSM [56] with an ensemble size of 7 models.
543 All other hyperparameters are the default values in the DreamerV2 repository.

544 The DreamerV2 baseline uses the same hyperparameters as used for MOTO (excluding the behav-
545 ior cloning factor and the disagreement penalty factor).

546 COMBO [12] and LOMPO [19] were run using the image-based implementations from the LOMPO
547 repository. For the image encoder network of the model, we use the default convolutional encoder
548 architecture, which has the following hyperparameters:

- 549 • channels: (32, 64, 128, 256)
- 550 • kernel sizes: (4, 4, 4, 4)
- 551 • strides: (2, 2, 2, 2)
- 552 • padding: VALID

553 • final MLP layer size: 1024

554 Similarly, the decoder network consists of Deconvolution/Transpose convolution layers with the
555 following hyperparameters:

556 • initial MLP layer size: 1024

557 • channels: (128, 64, 32, 3)

558 • kernel sizes: (5, 5, 6, 6)

559 • strides: (2, 2, 2, 2)

560 • padding: VALID

561 The latent dynamics model is represented using an RSSM [56] with an ensemble size of 7 models.

562 Both COMBO and LOMPO were trained using a model learning rate of 6×10^{-4} , and critic network
563 learning rate of 3×10^{-4} , and a policy network learning rate of 3×10^{-4} . The batch size for model
564 training is 64 and the batch size for agent training is 256. For COMBO, we use a conservatism
565 penalty factor of $\alpha = 2.5$, and for LOMPO we use a disagreement penalty factor of $\lambda = 5$.

566 **C.4 Model Free Methods**

567 The model free baselines (IQL [3], CQL [4], SAC [47], BC) were run using the JAXRL2 frame-
568 work [57]. For all policy networks, critic networks, and value networks, we used a feed-forward
569 convolutional encoder network architecture from the D4PG method [58], with the following hyper-
570 parameters:

571 • channels: (32, 64, 128, 256)

572 • kernel sizes: (3, 3, 3, 3)

573 • strides: (2, 2, 2, 2)

574 • padding: VALID

575 • final MLP layer size: 50

576 This encoder was then followed by two MLP layers of size 256, followed by a final output layer of
577 size 1 (for critic and value networks) or of size `action-dim` for policy networks. ReLU activations
578 were used between each layer.

579 We use a discount factor $\gamma = 0.99$ and a batch size of 256 for all of the methods, as well as a
580 learning rate of 3×10^{-4} for all policy, critic, and value networks. We also used a soft target update
581 for critic and value networks with a factor of $\tau = 0.005$. For CQL we set the conservatism penalty
582 factor $\alpha = 5$, and for IQL we set the expectile hyperparameter $\tau = 0.5$ and the inverse temperature
583 hyperparameter $\beta = 3$, which are the default values in JAXRL2. For all other hyperparameters, we
584 used the default values in JAXRL2.