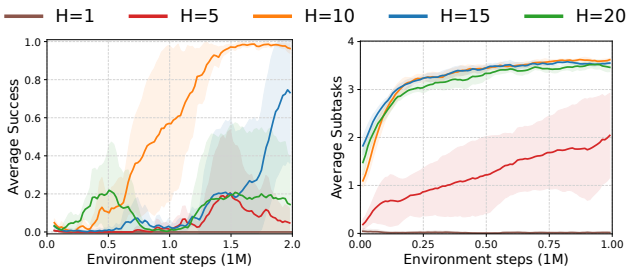
# A Further Ablations

We include additional ablations on the Maze and Kitchen tasks to further investigate the influence of skill horizon $H$, planning horizon $N$, and dynamics model fine-tuning, which is important for skill learning and planning.

## A.1 Skill Horizon

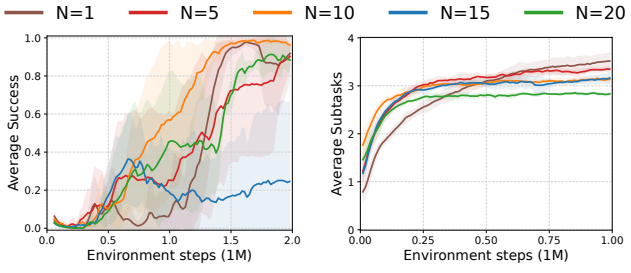



(a) Maze (b) Kitchen

Figure 6: Ablation analysis on skill horizon $H$.

In both Maze and Kitchen, we find that a too short skill horizon ($H = 1, 5$) is unable to yield sufficient temporal abstraction. A longer skill horizon ($H = 15, 20$) has little influence in Kitchen, but it makes the downstream performance much worse in Maze. This is because with long-horizon skills, a skill dynamics prediction becomes inaccurate and stochastic, and composing multiple skills can be not as flexible as short-horizon skills. The inaccurate skill dynamics makes long-term planning harder, which is already a major challenge in maze navigation.

## A.2 Planning Horizon

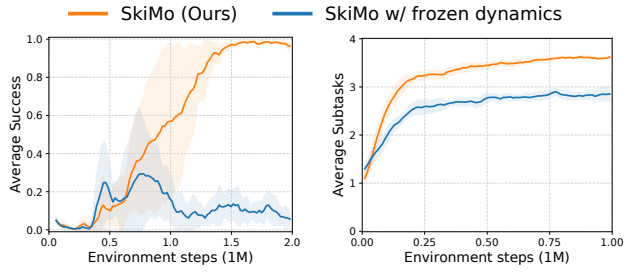



(a) Maze (b) Kitchen

Figure 7: Ablation analysis on planning horizon $N$.

In Figure 7b, we see that short planning horizon makes learning slower in the beginning, because it does not effectively leverage the skill dynamics model to plan further ahead. Conversely, if the planning horizon is too long, the performance becomes worse due to the difficulty in modeling every step accurately. Indeed, the planning horizon 20 corresponds to 200 low-level steps, while the episode length in Kitchen is 280, demanding the agent to make plan for nearly the entire episode. The performance is not sensitive to intermediate planning horizons. On the other hand, the effect of the planning horizon differs in Maze due to distinct environment characteristics. We find that very long planning horizon (eg. 20) and very short planning horizon (eg. 1) perform similarly in Maze (Figure 7a). This could attribute to the former creates useful long-horizon plans, while the latter avoids error accumulation altogether. We leave further investigation on planning horizon to future work.

## A.3 Fine-Tuning Model



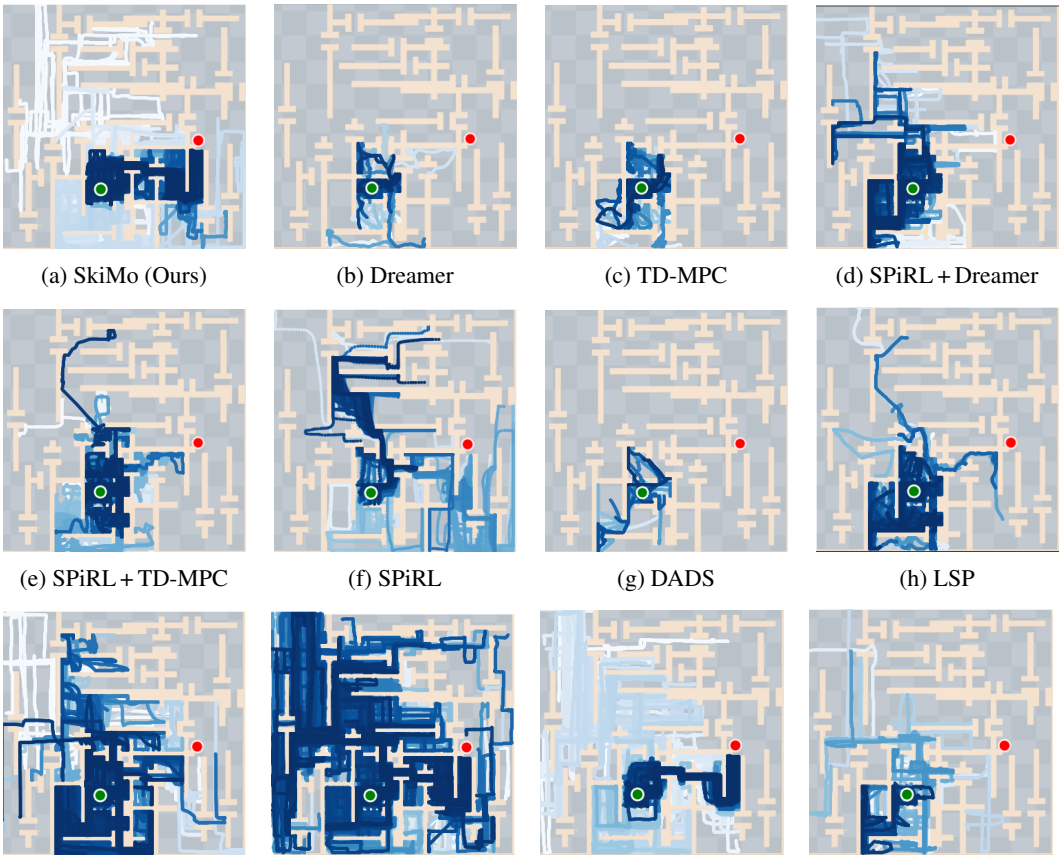

(a) Maze (b) Kitchen

Figure 8: Ablation analysis on fine-tuning the model.

We freeze the skill dynamics model together with the state encoder to gauge the effect of fine-tuning after pre-training. Figure 8 shows that without fine-tuning the model, the agent performs worse due to the discrepancy between distributions of the offline data and the downstream task. We hypothesize that fine-tuning is necessary when the agent needs to adapt to a different task and state distribution after pre-training.

# B Qualitative Analysis on Maze

## B.1 Exploration and Exploitation



(a) SkiMo (Ours) (b) Dreamer (c) TD-MPC (d) SPiRL + Dreamer

(e) SPiRL + TD-MPC (f) SPiRL (g) DADS (h) LSP

(i) SkiMo w/o joint training (j) SkiMo + SAC (k) SkiMo w/o CEM (l) SkiMo w/ single-step dynamics

Figure 9: Exploration and exploitation behaviors of our method and baseline approaches. We visualize trajectories in the replay buffer at 1.5M training steps in blue: light blue for early trajectories and dark blue for recent trajectories. Our method shows wide coverage of the maze at the early stage of training and fast convergence to the solution.

To gauge the agent's ability of exploration and exploitation, we visualize the replay buffer for each method in Figure 9. In this visualization, we represent early trajectories in the replay buffer with light blue dots and recent trajectories with dark blue dots. In Figure 9a, the replay buffer of *SkiMo* (ours) contains early explorations that span to most corners in the maze. After it finds the goal, it exploits this knowledge and commits to paths that are between the start location and the goal (in dark blue).

*Dreamer* and *TD-MPC* only explore a small fraction of the maze because they are prone to get stuck at walls without guided exploration from skills and skill priors. *SPiRL + Dreamer*, *SPiRL + TD-MPC*, and *SkiMo w/o joint training* explore better than *Dreamer* and *TD-MPC*, but all fail to find the goal. This is because without the joint training of the model and policy, the skill space is only optimized for action reconstruction, not for planning, which makes long-horizon exploration and exploitation harder.

On the other hand, *SkiMo + SAC* and *SPiRL* are able to explore the most portion of the maze, but even after the agent finds the goal through exploration, it continues to explore and does not exploit this experience to consistently accomplish the task (darker blue). This could attribute to the difficult long-horizon credit assignment problem which makes policy learning slow, and the reliance on skill prior which encourages exploration. On the contrary, our skill dynamics model effectively absorbs

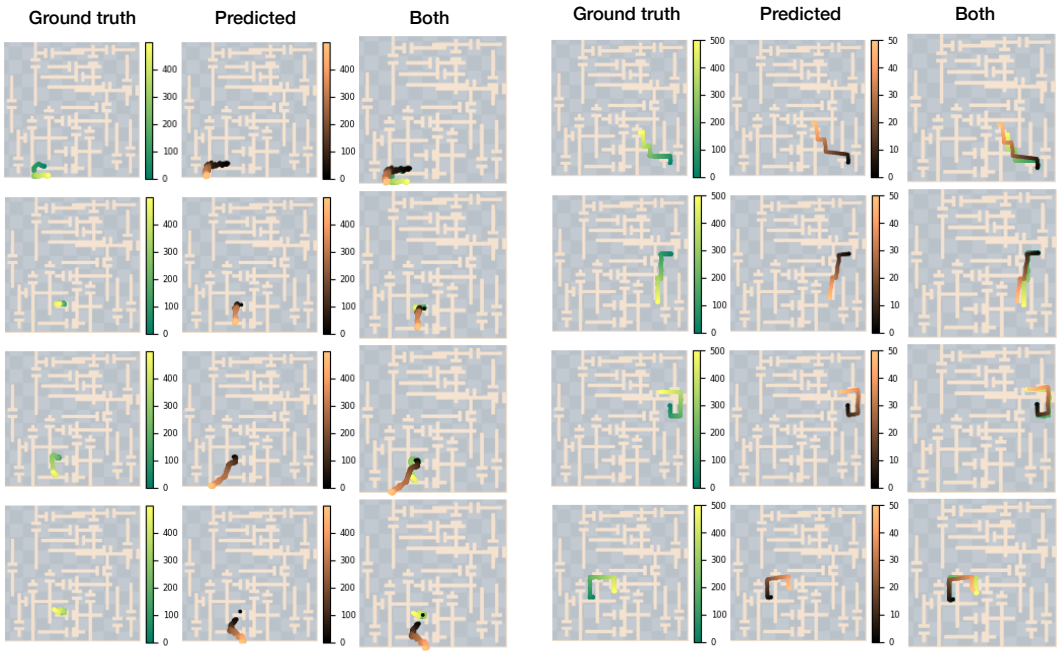
prior experience to generate goal-achieving imaginary rollouts for the actor and critic to learn from, which makes task learning more efficient.

We find the skill dynamics model useful in guiding the agent explore coherently and exploit efficiently. Without a temporally-extended model, *DADS*, *LSP*, and *SkiMo w/ single-step dynamics* fail to reach the goal. Even though they likewise condition on the skill latent, they still need to roll out the dynamics model step-by-step to predict future states. The single-step prediction is prone to compounding error for long-horizon planning. As a result, these agents do not collect sufficiently meaningful trajectories for the policy to learn. Additionally, *SkiMo w/o CEM* performs as well as *SkiMo*, indicating that CEM planning is not essential after the agent has already learned a good policy in Maze. Nevertheless, these qualitative results corroborate the effectiveness of our method.

### B.2 Long-horizon Prediction

To compare the long-term prediction ability of the skill dynamics and flat dynamics, we visualize imagined trajectories by sampling trajectory clips of 500 timesteps from the agent's replay buffer (the maximum episode length in Maze is 2,000), and predicting the latent state 500 steps ahead, which will be decoded using the observation decoder, given the initial state and 500 ground-truth actions (50 skills for *SkiMo*). The similarity between the imagined trajectory and the ground truth trajectory can indicate whether the model can make accurate predictions far into the future, producing useful imaginary rollouts for policy learning and planning.

*SkiMo* is able to reproduce the ground truth trajectory with little prediction error even when traversing through hallways and doorways while *Dreamer* struggles to make accurate long-horizon predictions due to error accumulation. This is mainly because *SkiMo* allows temporal abstraction in the dynamics model, thereby enabling temporally-extended prediction and reducing step-by-step prediction error.




(a) Dreamer (b) SkiMo (Ours)

Figure 10: Prediction results of 500 timesteps using a flat single-step model (a) and skill dynamics model (b), given the ground truth starting state and 500 actions (50 skills for *SkiMo*). The predicted states from the flat model deviate from the ground truth trajectory quickly while the prediction of our skill dynamics model has little error.

# C Implementation Details

## C.1 Computing Resources

Our approach and all baselines are implemented in PyTorch [46]. All experiments are conducted on a workstation with an Intel Xeon E5-2640 v4 CPU and a NVIDIA Titan Xp GPU. Pre-training of the skill policy and skill dynamics model takes around 10 hours. Downstream RL for 2M timesteps takes around 18 hours. The policy and model update frequency is the same over all algorithms but Dreamer [3] and TD-MPC [4]. Since Dreamer and TD-MPC train on primitive actions, it has 10 times more frequent model and policy updates than skill-based algorithms, which leads to slower training (about 52 hours).

## C.2 Algorithm Implementation Details

For the baseline implementations, we use the official code for SPiRL, DADS, and LSP. We re-implemented Dreamer and TD-MPC in PyTorch, which are verified on DeepMind Control Suite [47]. The table below (Table 1) compares key components of *SkiMo* with model-based and skill-based baselines and ablated methods.

Table 1: Comparison to prior work and ablated methods.

| Method | Skill-based | Model-based | Joint training |
|---|---|---|---|
| Dreamer [3] and TD-MPC [4] | ✗ | ✓ | ✗ |
| DADS [29] and LSP [36] | ✓ | ✓ | ✗ |
| SPiRL [12] | ✓ | ✗ | ✗ |
| SPiRL + Dreamer and SPiRL + TD-MPC | ✓ | ✓ | ✗ |
| SkiMo w/o joint training | ✓ | ✓ | ✗ |
| SkiMo + SAC | ✓ | ✗ | ✓ |
| SkiMo (Ours) and SkiMo w/o CEM | ✓ | ✓ | ✓ |

**Dreamer [3]** We use the same hyperparameters with the official implementation.

**TD-MPC [4]** We use the same hyperparameters with the official implementation, except that we do not use the prioritized experience replay [48]. The same implementation is used for the SPiRL + TD-MPC baseline and our method with only minor modification.

**SPiRL [12]** We use the official implementation of the original paper and use the hyperparameters suggested in the official implementation.

**SPiRL + Dreamer [12]** We use our implementation of Dreamer and simply replace the action space with the latent skill space of SPiRL. We use the same pre-trained SPiRL skill policy and skill prior networks with the SPiRL baseline. Initializing the high-level downstream task policy with the skill prior, which is critical for downstream learning performance [12], is not possible due to the policy network architecture mismatch between Dreamer and SPiRL. Thus, we only use the prior divergence to regularize the high-level policy instead. Directly pre-train the high-level policy did not lead to better performance, but it might have worked better with more tuning.

**SPiRL + TD-MPC [4]** Similar to SPiRL + Dreamer, we use our implementation of TD-MPC and replace the action space with the latent skill space of SPiRL. The initialization of the task policy is also not available due to the different architecture used for TD-MPC.

**DADS [29]** We use the official implementation and hyperparameters of the original paper, except that we use DADS on a sparse reward setup since dense reward is not available in our tasks.

**LSP [36]** We use the code provided by the authors and the default hyperparameters in the code.

**SkiMo (Ours)** The skill-based RL part of our method is inspired by Pertsch et al. [12] and the model-based component is inspired by Hansen et al. [4] and Hafner et al. [3]. We elaborate our skill and skill dynamics learning in Algorithm 1, planning algorithm in Algorithm 2, and model-based RL in Algorithm 3. Table 2 lists the all hyperparameters that we used.

**Algorithm 1** SkiMo (*skill and skill dynamics learning*)

**Require:** $\mathcal{D}$: offline task-agnostic data
Randomly initialize $\theta, \psi$
$\psi^- \leftarrow \psi$ ▷ initialize target network
**for** each iteration **do**
  Sample mini-batch $B = (\mathbf{s}, \mathbf{a})_{(0:NH)} \sim \mathcal{D}$
  $[\theta, \psi] \leftarrow [\theta, \psi] - \lambda_{[\theta,\psi]} \nabla_{[\theta,\psi]} \mathcal{L}(B)$ ▷ $\mathcal{L}$ from Equation (5)
  $\psi^- \leftarrow (1-\tau)\psi^- + \tau\psi$ ▷ update target network
**end for**
**return** $\theta, \psi, \psi^-$

**Algorithm 2** SkiMo (*CEM planning*)

**Require:** $\theta, \psi, \phi$ : learned parameters, $\mathbf{s}_t$: current state
$\mu^0, \sigma^0 \leftarrow \mathbf{0}, \mathbf{1}$ ▷ initialize sampling distribution
**for** $i = 1, ..., N_{\text{CEM}}$ **do**
  Sample $N_{\text{sample}}$ trajectories of length $N$ from $\mathcal{N}\big(\mu^{i-1}, (\sigma^{i-1})^2\big)$ ▷ sample skill sequences from normal distribution
  Sample $N_\pi$ trajectories of length $N$ using $\pi_\phi, D_\psi$ ▷ sample skill sequences via imaginary rollouts
  Estimate $N$-step returns of $N_{\text{sample}} + N_\pi$ trajectories using $R_\phi, Q_\phi$
  Compute $\mu^i, \sigma^i$ with top-k return trajectories ▷ update parameters for next iteration
**end for**
Sample a skill $\mathbf{z} \sim \mathcal{N}\big(\mu^{N_{\text{CEM}}}, (\sigma^{N_{\text{CEM}}})^2\big)$
**return** $\mathbf{z}$

**Algorithm 3** SkiMo (*downstream RL*)

**Require:** $\theta, \psi, \psi^-$ : pre-trained parameters
$\mathcal{B} \leftarrow \emptyset$ ▷ initialize replay buffer
Randomly initialize $\phi$
$\phi^- \leftarrow \phi$ ▷ initialize target network
$\pi_\phi \leftarrow p_\theta$ ▷ initialize task policy with skill prior
**for** not converged **do**
  $t \leftarrow 0, s_0 \sim \rho_0$ ▷ initialize episode
  **for** episode not done **do**
    $\mathbf{z}_t \sim \text{CEM}(\mathbf{s}_t)$ ▷ MPC with CEM planning in Algorithm 2
    $\mathbf{s}, r_t \leftarrow \mathbf{s}_t, 0$
    **for** $H$ steps **do**
      $\mathbf{s}, r \leftarrow \text{ENV}(\mathbf{s}, \pi_\theta^L(E_\psi(\mathbf{s}), \mathbf{z}_t))$ ▷ rollout low-level skill policy
      $r_t \leftarrow r_t + r$
    **end for**
    $\mathcal{B} \leftarrow \mathcal{B} \cup (\mathbf{s}_t, \mathbf{z}_t, r_t)$ ▷ collect $H$-step environment interaction
    $t \leftarrow t + H$
    $\mathbf{s}_t \leftarrow \mathbf{s}$
    Sample mini-batch $B = (\mathbf{s}, \mathbf{z}, r)_{(0:N)} \sim \mathcal{B}$
    $[\phi, \psi] \leftarrow [\phi, \psi] - \lambda_{[\phi,\psi]} \nabla_{[\phi,\psi]} \mathcal{L}'_{\text{REC}}(B)$ ▷ $\mathcal{L}'_{\text{REC}}$ from Equation (6)
    $\phi_\pi \leftarrow \phi_\pi - \lambda_\phi \nabla_{\phi_\pi} \mathcal{L}_{\text{RL}}(B)$ ▷ $\mathcal{L}_{\text{RL}}$ from Equation (7). Update only policy parameters
    $\psi^- \leftarrow (1-\tau)\psi^- + \tau\psi$ ▷ update target network
    $\phi^- \leftarrow (1-\tau)\phi^- + \tau\phi$ ▷ update target network
  **end for**
**end for**
**return** $\psi, \phi$

### C.3 Environments and Offline Data

**Maze [43, 16]** Since our goal is to leverage offline data collected from diverse tasks in *the same environment*, we use a variant of the D4RL maze environment [43], suggested in Pertsch et al. [16]. The maze is of size $40 \times 40$; an initial state is randomly sampled near a pre-defined region (the green circle in Figure 3a); and the goal is fixed shown as the red circle in Figure 3a. The observation consists of the agent's 2D position and velocity. The agent moves around the maze by controlling the continuous value of its $(x, y)$ velocity. The maximum episode length is 2,000 but an episode is also terminated if the agent reaches the circle around the goal with radius 2. The reward of 100 is given at task completion. We use the offline data from Pertsch et al. [16], consisting of 3,046 trajectories with randomly sampled start and goal state pairs. Thus, the offline data and downstream task share the same environment, but have different start and goal states (i.e. different tasks). This data can be used to extract short-horizon skills like navigating hallways or passing through narrow doors.

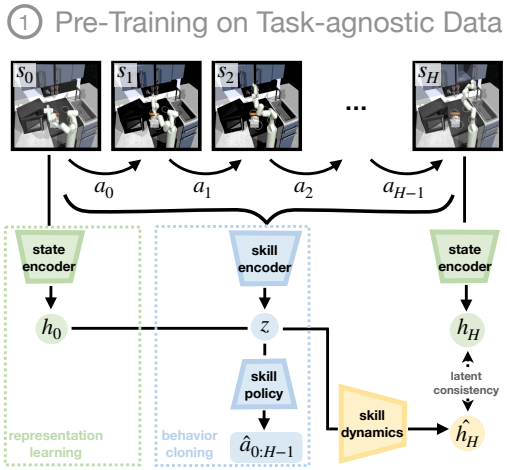
**Kitchen [32, 43]** The 7-DoF Franka Emika Panda robot arm needs to perform four sequential tasks: open microwave, move kettle, turn on bottom burner, and flip light switch. The agent has a 30D observation space (11D robot proprioceptive state and 19D object states), which removes a constant 30D goal state in the original environment, and 9D action space (7D joint velocity and 2D gripper velocity). The agent receives a reward of 1 for every sub-task completion. The episode length is 280 and an episode also ends once all four sub-tasks are completed. The initial state is set with a small noise in every state dimension. We use 603 trajectories collected by teleoperation from Gupta et al. [32] as the offline task-agnostic data. The data involves interaction with all seven manipulatable objects in the environment, but during downstream learning the agent needs to execute an unseen sequence of four subtasks. Thus, the agent can transfer a rich set of manipulation skills, but needs to recombine them in new ways to solve the task.

**Mis-aligned Kitchen [16]** The environment and task-agnostic data are the same with **Kitchen** but we use the different downstream task: open microwave, flip light switch, slide cabinet door, and open hinge cabinet, as illustrated in Figure 3c. This task ordering is not aligned with the sub-task transition probabilities of the task-agnostic data, which leads to challenging exploration following the prior from data. This is because the transition probabilities in the Kitchen human-teleoperated dataset are not uniformly distributed; instead, certain transitions are more likely than others. For example, the first transition in our target task — from opening the microwave to flipping the light switch — is very unlikely to be observed in the training data. This simulates the real-world scenario where the large offline dataset may not be meticulously curated for the target task.

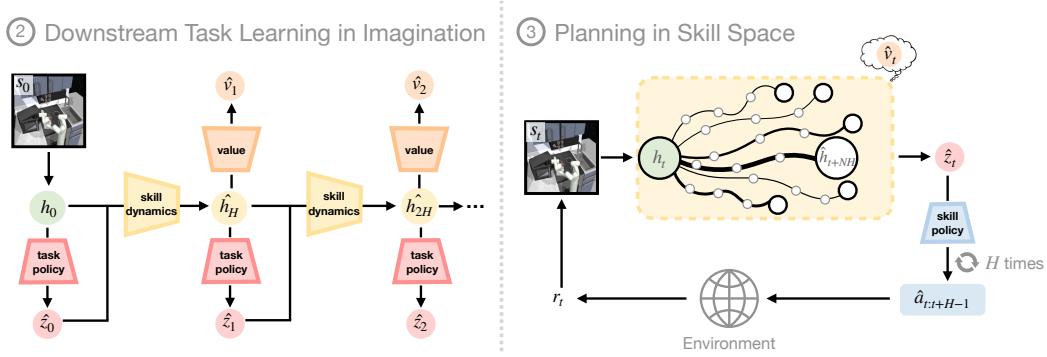
**CALVIN [44]** We adapt the CALVIN environment [44] for long-horizon learning with the state observation. The CALVIN environment uses a Franka Emika Panda robot arm with 7D end-effector pose control (relative 3D position, 3D orientation, 1D gripper action). The 21D observation space consists of the 15D proprioceptive robot state and 6D object state. We use the teleoperated play data (Task D→Task D) of 1,239 trajectories from Mees et al. [44] as our task-agnostic data. The agent receives a sparse reward of 1 for every sub-task completion in the correct order: open drawer, turn on lightbulb, move slider left, and turn on LED. The episode length is 360 and an episode also ends if all four sub-tasks are completed. In data, there exist 34 available target sub-tasks, and each sub-task can transition to any other sub-task, which makes any transition probability lower than 0.1% on average.

## D Application to Real Robot Systems

Our algorithm is designed to be applied on real robot systems by improving sample efficiency of RL using a temporally-abstracted dynamics model. Throughout the extensive experiments in simulated robotic manipulation environments, we show that our approach achieves superior sample efficiency over prior skill-based and model-based RL, which gives us strong evidence for the application to real robot systems. Especially in Kitchen and CALVIN, our approach improves the sample efficiency of learning long-horizon manipulation tasks with a 7-DoF Franka Emika Panda robot arm. Our approach consists of three phases: (1) task-agnostic data collection, (2) skills and skill dynamics model learning, and (3) downstream task learning. In each of these phases, our approach can be applied to physical robots:




(a) In pretraining, *SkiMo* leverages offline task-agnostic data to extract skill dynamics and a skill repertoire. Unlike prior works that keep the model and skill policy training separate, we propose to *jointly* train them to extract a skill space that is conducive to plan upon.




(b) In downstream RL, we learn a high-level task policy in the skill space (skill-based RL) and leverage the skill dynamics model to generate imaginary rollouts for policy optimization and planning (model-based RL).

Figure 11: Illustration of our algorithm, SkiMo.

**Task-agnostic data collection** Our approach is designed to fully leverage task-agnostic data without any reward or task annotation. In addition to extracting skills and skill priors, we further learn a skill dynamics model from this task-agnostic data. Maximizing the utility of task-agnostic data is critical for real robot systems as data collection with physical robots itself is very expensive. Our method does not require any manual labelling of data and simply extracts skills, skill priors, and skill dynamics model from raw states and actions, which makes our method scalable.

**Pre-training of skills and skill dynamics model** Our approach trains the skill policy, skill dynamics model, and skill prior from the offline task-agnostic dataset, without requiring any additional real-world robot interactions.

**Downstream task learning** The goal of our work is to leverage skills and skill dynamics model to allow for more efficient downstream learning, i.e., requires less interactions of the agent with the environment for training the policy. This is especially important on real robot systems where a robot-environment interaction is slow, dangerous, and costly. Our approach directly addresses this concern by learning a policy from imaginary rollouts rather than actual environment interactions.

In summary, we believe that SkiMo can be applied to real-world robot systems with only minor modifications.

Table 2: SkiMo hyperparameters.

| Hyperparameter | Value | | |
|---|---|---|---|
| | Maze | FrankaKitchen | CALVIN |
| **Model architecture** | | | |
| # Layers of $O_\theta, p_\theta, \pi_\theta^L, E_\psi, D_\psi, \pi_\phi, R_\phi, Q_\phi$ | | 5 | |
| Activation funtion | | elu | |
| Hidden dimension | 128 | 128 | 256 |
| State embedding dimension | 128 | 256 | 256 |
| Skill encoder ($q_\theta$) | 5-layer MLP | LSTM | LSTM |
| Skill encoder hidden dimension | | 128 | |
| **Pre-training** | | | |
| Pre-training batch size | | 512 | |
| # Training mini-batches per update | | 5 | |
| Model-Actor joint learning rate ($\lambda_{[\theta,\psi]}$) | | 0.001 | |
| Encoder KL regularization ($\beta$) | | 0.0001 | |
| Reconstruction loss coefficient ($\lambda_{\text{O}}$) | | 1 | |
| Consistency loss coefficient ($\lambda_{\text{L}}$) | | 2 | |
| Low-level actor loss coefficient ($\lambda_{\text{BC}}$) | | 2 | |
| Planning discount ($\rho$) | | 0.5 | |
| Skill prior loss coefficient ($\lambda_{\text{SP}}$) | | 1 | |
| **Downstream RL** | | | |
| Model learning rate | | 0.001 | |
| Actor learning rate | | 0.001 | |
| Skill dimension | | 10 | |
| Skill horizon ($H$) | | 10 | |
| Planning horizon ($N$) | 10 | 3 | 1 |
| Batch size | 128 | 256 | 256 |
| # Training mini-batches per update | | 10 | |
| State normalization | True | False | False |
| Prior divergence coefficient ($\alpha$) | 1 | 0.5 | 0.1 |
| Alpha learning rate | 0.0003 | 0 | 0 |
| Target divergence | 3 | N/A | N/A |
| # Warm up step | 50,000 | 5,000 | 5,000 |
| # Environment step per update | | 500 | |
| Replay buffer size | | 1,000,000 | |
| Target update frequency | | 2 | |
| Target update tau ($\tau$) | | 0.01 | |
| Discount factor ($\gamma$) | | 0.99 | |
| Reward loss coefficient ($\lambda_{\text{R}}$) | | 0.5 | |
| Value loss coefficient ($\lambda_{\text{Q}}$) | | 0.1 | |
| **CEM** | | | |
| CEM iteration ($N_{\text{CEM}}$) | | 6 | |
| # Sampled trajectories ($N_{\text{sampled}}$) | | 512 | |
| # Policy trajectories ($N_\pi$) | | 25 | |
| # Elites ($k$) | | 64 | |
| CEM momentum | | 0.1 | |
| CEM temperature | | 0.5 | |
| Maximum std | | 0.5 | |
| Minimum std | | 0.01 | |
| Std decay step | 100,000 | 25,000 | 25,000 |
| Horizon decay step | 100,000 | 25,000 | 25,000 |