Dynamics Learning with Cascaded Variational Inference for Multi-Step Manipulation

Kuan Fang[1], Yuke Zhu[1,2], Animesh Garg[2,3], Silvio Savarese[1], Li Fei-Fei[1]
[1] Stanford University, [2] Nvidia, [3] University of Toronto & Vector Institute

Abstract: The fundamental challenge of planning for multi-step manipulation is to find effective and plausible action sequences that lead to the task goal. We present Cascaded Variational Inference Planner (CAVIN), a model-based method that hierarchically generates plans by sampling from latent spaces. To facilitate planning over long time horizons, our method learns latent representations that decouple the prediction of high-level effects from the generation of low-level motions through cascaded variational inference. This enables us to model dynamics at two different levels of temporal resolutions for hierarchical planning. We evaluate our approach in three multi-step robotic manipulation tasks in cluttered tabletop environments given raw visual observations. Empirical results demonstrate that the proposed method outperforms state-of-the-art model-based approaches by strategically planning for interactions with multiple objects. See more details at
`pair.stanford.edu/cavin`

Keywords: dynamics modeling, latent-space planning, variational inference

1 Introduction

Sequential manipulation planning is a hallmark of intelligence. Many animal species have demonstrated remarkable abilities to perform multi-step tasks [1, 2]. Nonetheless, the ability to solve multi-step manipulation problems remains an open challenge for today's robotic research. The challenge involves high-level reasoning about what are the desired states to reach, as well as low-level reasoning about how to execute actions to arrive at these states. Therefore, an effective algorithm should not only make a high-level plan which describes desired effects during task execution, but also produce feasible actions under physical and semantic constraints of the environment.

Conventional methods have formulated this as the task and motion planning problem [3, 4]. However, the applicability of these methods has been hindered by the uncertainty raised from raw visual perception in unstructured environments. To solve multiple tasks in visual environments in a data-efficient manner, model-based approaches, powered by deep neural networks, have been proposed to use data-driven dynamics models for planning [5, 6, 7]. Given the trajectories predicted by the dynamics model, a plan can be generated through sampling algorithms, such as uniform sampling and cross entropy method [8]. These methods have shown successes in many control tasks given visual input, such as playing video games [5], pushing a specified object in a cluttered tray [7], and manipulating deformable objects [9].

However, naïve sampling approaches suffer from the curse of dimensionality when handling the large sampling space in manipulation domains. In realistic problems, we have to deal with continuous (and often high-dimensional) state space and action spaces and long task horizons, whereas a small fraction of actions is valid and effective and a small subset of state sequences could lead to high rewards. To boost sampling efficiency, recent work has proposed to use generative models [10, 11, 12] to prioritize the sampling of more promising states and actions. These works do not exploit the hierarchical structure of multi-step tasks but rather making a flat plan on the low-level actions. As a result, the methods have mostly focused on short-horizon tasks.

Our key insight for effective planning in multi-step manipulation tasks is to take advantage of the hierarchical structure of the action space, such that the generation of an action sequence can be factorized into a two-level process: 1) generating a high-level plan that describes the desired effects in the state space, and 2) generating a low-level plan of motions to produce the desired effects. To this
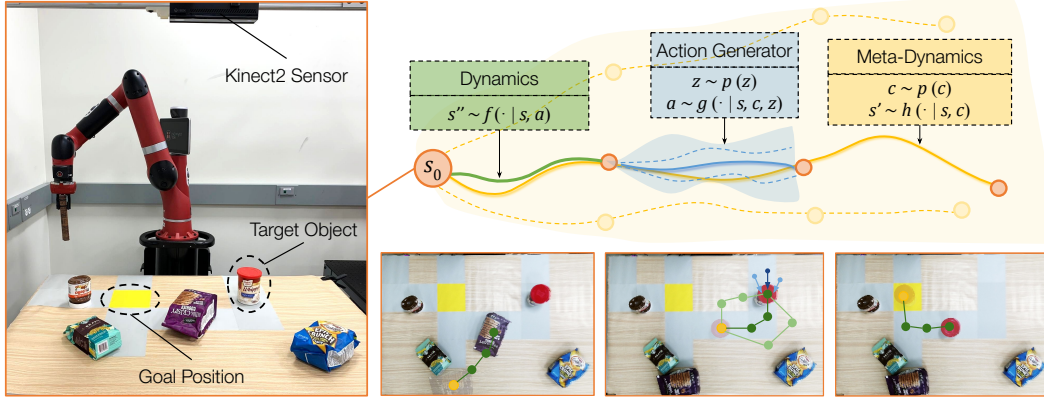
Figure 1: Hierarchical planning in latent spaces for multi-step manipulation tasks. The manipulation tasks shown in the figure requires the robot to move the target object to a goal position through specified regions (marked by grey tiles). In presence of an obstacle, the planner needs to move the obstacles aside and then move the target. We propose to use three tightly coupled modules: dynamics model, meta-dynamics model and action generator (see details in Sec. 3) to hierarchically generate plans for the task goal. Planning in learned latent spaces, our method first predicts subgoals (yellow) and then generates plausible actions (blue). The optimal plan is chosen by predicting resultant state trajectories (green) of the sampled actions. The selected plan is in darker colors.

end, we propose Cascaded Variational Inference Planner (CAVIN) to produce plans from plannable latent representations of effects and motions. As illustrated in Fig. 1, CAVIN generates a high-level plan of desired effects predicted at a coarse temporal resolution and a low-level plan of motions at a fine-grained resolution. By decoupling effects and motions, the proposed method substantially reduces the search space of the optimal plan, enabling CAVIN to solve longer-horizon manipulation tasks with high-dimensional visual states and continuous actions. Our approach jointly learns the effect-level plans and motion-level plans, as opposed to learning the motions in a post-hoc manner in previous work [13]. To achieve this, we propose a cascaded generative model to jointly capture the distribution of actions and resultant states. We employ variational principles [14] in a cascaded manner to derive lower bound objectives for learning from task-independent self-supervision.

Our contributions of this work are three-fold:

1. We introduce a cascaded variational inference framework to learn latent representations for hierarchical planning. Our method decouples the high-level prediction of effects from the low-level generation of motions for multi-step manipulation through learned latent spaces

2. We introduce a model-based planner that takes advantage of the learned latent presentations. By performing predictive control (MPC) at two temporal resolutions, the proposed planner substantially reduces the computational burden of sampling continuous actions over long time horizons.

3. We showcase three multi-step robotic manipulation tasks in simulation and the real world. These tasks involve a robotic arm interacting with multiple objects in a cluttered environment for achieving predefined task goals. We compare the proposed approach with the state-of-the-art baselines. Empirical results demonstrate that our hierarchical modeling improves performance in all three tasks.

2   Preliminaries

We consider robotic manipulation in an environment with unknown dynamics. Each manipulation task in the environment can be formulated as a Markov Decision Process (MDP) with high-dimensional state space $\mathcal{S}$ and continuous action space $\mathcal{A}$. We denote the dynamics as $p : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, where $p(s_{t+1}|s_t, a_t)$ is the transition probability from s_t to s_{t+1} by taking action a_t. In our tabletop manipulation tasks, s_t is the processed visual observations of the workspace from RGB-D cameras and a_t is the control command of the robot (see Sec. 4 for details). Our objective is to find a sequence of actions $a_{1:H}$ to maximize the cumulative reward $\mathbb{E}_\pi[\sum_t R(s_t, a_t, s_{t+1})]$ across planning horizon H, where $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the given immediate reward function. In this work, we consider the immediate reward function in the form of $R(s_{t+1})$ which can be directly evaluated from the next state. This enables planning in the state space while abstracting away the actions.

We address the problem of finding the optimal action sequence $a^*_{1:H}$ with model-based planning. At the core of such an approach is a model predictive control (MPC) algorithm using a dynamics model of the environment [15]. Given the known or learned dynamics model, we can predict resultant states of sampled actions by rolling out the actions with the dynamics model. The robot receives new observed state s_t from the environment every T steps and plans for the action sequence $a_{t:t+T-1}$. The planned actions are executed in a closed-loop manner for T steps before the new state is received.

3 Method

We present a hierarchical planning algorithm learned with Cascaded Variational Inference (CAVIN) to facilitate long-horizon, multi-step manipulation tasks. Our key insight is to learn latent spaces that facilitate planning by leveraging the hierarchical structure of the action space, such that the generation of an action sequence can be factorized into a two-level process: 1) generating a high-level plan that describes the *effects* to achieve in the state space, and 2) generating a low-level plan of *motions* that produces the desired effects.

3.1 Cascaded Generative Process of Dynamics

We propose to use a cascaded generative process to capture the distribution of actions and resultant states for efficient sampling-based planning. Instead of sampling in the original action space $\mathcal{A}$, we construct two latent spaces to factorize the generation of actions to prioritize promising samples. We define $\mathcal{C}$ as the latent effect space, where an **effect code** $c \in \mathcal{C}$ describes the desired effect of the action, and $\mathcal{Z}$ as the latent motion space, where a **motion code** $z \in \mathcal{Z}$ defines the detailed motion of the action. Intuitively, each effect code represents a reachable subgoal from the current state, while each motion code represents a distinctive action sequence that leads to the subgoal. In our model, both c and z are vector continuous variables with priors as standard normal distribution $\mathcal{N}(0, I)$.

Dynamics of the environment can be defined on two levels by introducing three tightly-coupled model components as shown in Fig. 1. On the low-level, we use the **dynamics model** f to estimate the transition probability in the original state and action spaces. On the high-level, we use the **meta-dynamics model** h which characterizes the distribution of the subgoal state s_{t+T} that is T steps away from the current state s_t conditioned on the effect code c_t. Compared to the low-level dynamics model, the meta-dynamics model operates at a coarser temporal resolution T and abstracts away the detailed actions, which enables effective hierarchical planning in long-horizon tasks. The two levels of dynamics are related by the **action generator** g which characterizes the distribution of the action sequence $a_{t:t+T-1}$ given s_t and conditioned on a motion code z_t and the targeted subgoals represented by the effect code c_t. Given the same effect code c_t at state s_t, the meta-dynamics model and the action generator are supposed to produce consistent outputs, in the sense that by taking action sequence $a_{t:t+T-1}$ generated by the action generator the environment should ideally reach the subgoal state s_{t+T} computed by the meta-dynamics model after T steps.

Formally, the cascaded generative process of the two-level dynamics can be defined as:

$$s_{t+1} \quad \sim \quad f(\cdot|s_t, a_t; \theta_f) \tag{1}$$
$$s_{t+T} \quad \sim \quad h(\cdot|s_t, c_t; \theta_h) \tag{2}$$
$$a_{t:t+T-1} \quad \sim \quad g(\cdot|s_t, c_t, z_t; \theta_g) \tag{3}$$

where f, h and g are Gaussian likelihood functions parameterized by nonlinear deep networks θ_f, θ_h and θ_g respectively. We implement the modules using relational networks [16].

3.2 Hierarchical Planning in Latent Spaces

The hierarchical planning algorithm in the latent spaces $\mathcal{C}$ and $\mathcal{Z}$ is shown in Algorithm 1. Every T steps, the planning algorithm receives the current state s_t and plans across the planning horizon H. Without loss of generality, let H be divisible by T. The action sequence $a_{t:t+T-1}$ is produced by the algorithm and executed in the environment in a closed-loop manner. To effectively generate plausible plans, the planning is performed in the latent spaces instead of the original action space by hierarchically finding the optimal effect and motion codes. The optimal action sequence $a^*_{t:t+T-1}$ is then computed using the action generator described in Equation (3) given the selected codes.

Effect-Level Plan. On the high-level, we plan for subgoals towards reaching the final task goal by choosing the optimal sequence of effect codes $c^*_{1:K}$, where $K = T/H$ is the number of subgoals.

3

Algorithm 1 Hierarchical Planning with Cascaded Generative Processes

---

Require: initial state s_0, planning horizon H, temporal abstraction T, number of plan samples N

1: $K = H/T$
2: $t = 0$
3: **while** episode not done **do**
4:     Receive the new s_t from the environment.
5:     Sample N sequences of effect codes of length K as $\{c_{1:K}^i\}_{i=1}^N$.
6:     Predict subgoals $\{s_{t+T}^i, s_{t+2T}^i, \dots, s_{t+KT}^i\}_{i=1}^N$ by recursively using Equation (2) with
         $\{c_{1:K}^i\}_{i=1}^N$ and interpolate the state trajectory.
7:     Choose $c_{1:K}^*$ and corresponding subgoals with the highest cumulative reward $\sum_{t=1}^T R(s_t)$.
8:     Sample N sequences of motion codes of length K as $\{z_{1:K}^j\}_{j=1}^N$.
9:     Predict sequences of actions $\{a_t^j, \dots, a_{t+H-1}^j\}_{j=1}^N$ and states $\{s_{t+1}^j, \dots, s_{t+H}^j\}_{j=1}^N$ with
         $c_{1:K}^*$ and $\{z_{1:K}^j\}_{j=1}^N$ by recursively using Equation (3) and (1).
10:     Choose $z_{1:K}^*$ and corresponding $a_{t:t+H-1}^*$ which lead to states closest to the chosen subgoals.
11:     Execute $a_{t:t+T-1}^*$ for T steps.
12:     Update $t = t + T$.
13: **end while**

N sequences of effect codes are sampled from the prior probability $p(c)$, which we denote as $\{c_{1:K}^i\}_{i=1}^N$. Using Equation (2), the trajectory of subgoals corresponds to each $c_{1:K}^i$ are predicted as $s_{t+T}^i, s_{t+2T}^i, \dots, s_{t+KT}^i$. Between every two adjacent subgoals, we linearly interpolate the states of each time step and evaluate the immediate reward $R(s_t)$. We use the cumulative rewards to rank the predicted sequence of states. The highest-ranked sequence $c_{1:K}^*$ and the corresponding subgoals are selected to serve as intermediate goals in the low-level planning.

Motion-Level Plan. On the low-level, we generate the sequence of actions in the context of the desired effects indicated by c_t, by choosing the optimal sequence of motion codes $z_{1:K}^*$. N sequences of motion codes are sampled from the prior probability $p(z)$ as $\{z_{1:K}^j\}_{j=1}^N$. Each sequence is paired with the selected $c_{1:K}^*$ to produce the action sequence. Each pair of effect and motion codes is projected to a segment of action sequence of T steps using the action generator g as in Equation (3). Then the resultant state trajectories are predicted by the dynamics model f using Equation (1). We choose $z_{1:K}^*$ which lead to states that are closest to the chosen subgoals in the high-level planning.

The optimal action sequence $a_{1:H}^*$ is produced as the one that corresponds to the selected $c_{1:K}^*$ and $z_{1:K}^*$. The first T planned actions are provided to the robot for execution. Re-planning occurs every T steps when the new state is received.

3.3   Learning with Cascaded Variational Inference

The dynamics model f, meta-dynamics model h and action generator g are learned with respect to parameters θ_h, θ_f and θ_g to fit on transition data $\mathcal{D} = \{(s_t^i, a_t^i, s_{t+1}^i)\}_{i=1}^M$ observed from the environment. $\mathcal{D}$ can be collected either by self-supervision from the robot's own experiences or human demonstrations. In this paper, we use a heuristic policy that randomly samples actions that are likely to have plausible effects to the environment. The learning algorithm aims to maximize the marginal likelihoods $p(s_{t+1}|s_t, a_t; \theta_f)$, $p(s_{t+T}|s_t; \theta_h)$, and $p(a_{t:t+T-1}|s_t, c_t; \theta_g)$ on the dataset $\mathcal{D}$ under the entire generative process as described in Sec.3.1.

For the low-level dynamics, the likelihood is directly maximized by observed tuples of s_t, a_t and s_{t+1} from $\mathcal{D}$. We define $f(s_{t+1}|s_t, a_t; \theta_f)$ as a function to predict the mean of s_{t+1} with a fixed covariance matrix, so maximizing the likelihood is equivalent to minimizing a reconstruction loss between the observed s_{t+1} and the prediction.

The meta-dynamics model and the action generator are trained with transition sequences of consecutive T steps from $\mathcal{D}$. As the effect code c and the motion code z are latent variables, the likelihoods $p(s_{t+T}|s_t; \theta_h)$ and $p(a_{t:t+T-1}|s_t, c_t; \theta_g)$ cannot be directly maximized. Instead, we follow the variational principle [14] to derive a lower bound objective on these two marginal likelihoods by constructing inference models of c and z. For succinctness, we drop the subscripts from the symbols in the following equations using s' to denote the next subgoal state s_{t+T}, s'' to denote the next state s_{t+1}, and a to denote the action sequence $a_{t:t+T-1}$.

Lower Bound Objective. We construct inference models which consist of approximate posterior distributions $q_h(c|s, s'; \phi_h)$ and $q_g(z|s, a, c; \phi_g)$ both as neural networks with trainable parameters denoted as ϕ_h and ϕ_g. Thus the evidence lower bound objective (ELBO) [14] for marginal likelihoods can be derived as below.

For the meta-dynamics h, the variational bound $\mathcal{J}_h$ on the marginal likelihood for a single transition is defined to be a standard CVAE [14] conditioned on s as:

$$\log p(s'|s; \theta_h) \geq \mathbb{E}_{q_h(c|s,s';\phi_h)}[\log h(s'|s, c; \theta_h)] - D_{KL}(q_h(c|s, s'; \phi_h) \parallel p(c)) = -\mathcal{J}_h \quad (4)$$

For the action generator g, directly maximizing $p(a|s, c; \theta_g)$ conditioned on the unobserved c is intractable. Instead, we maximize $p(a|s, s'; \theta_g)$ by marginalizing over the inferred c given observed transitions from s to s':

$$p(a|s, s'; \theta_g) = \mathbb{E}_{q_h(c|s,s';\phi_h)}[p(a|s, s', c; \theta_g)] = \mathbb{E}_{q_h(c|s,s';\phi_h)}[p(a|s, c; \theta_g)] \quad (5)$$

Assume c is given, the variational bound $\mathcal{J}_{g|c}$ of the marginal likelihood $p(a|s, c; \theta_g)$ is:

$$\log p(a|s, c; \theta_g) \geq \mathbb{E}_{q_g(z|s,a,c;\phi_g)}[\log g(a|s, c, z; \theta_g)] - D_{KL}(q_g(z|s, a, c; \phi_g) \parallel p(z)) = -\mathcal{J}_{g|c}$$
$$(6)$$

Thus the variational lower bound $\mathcal{J}_g$ of marginal likelihood $p(a|s, s')$ can be derived as:

$$p(a|s, s'; \theta_g) = \mathbb{E}_{q_h(c|s,s';\phi_h)}[p(a|s, c; \theta_g)] \geq -\mathbb{E}_{q_h(c|s,s';\phi_h)}[\mathcal{J}_{g|c}] = -\mathcal{J}_g \quad (7)$$

The maximum likelihood estimation of $p(s'|s; \theta_h)$ and $p(a|s, c; \theta_g)$ now becomes minimizing the objective function $\mathcal{J}_h + \mathcal{J}_g$ on $\mathcal{D}$ with respect to parameters $\theta_h, \theta_g, \phi_h$ and ϕ_g. All these parameters are trained end-to-end using gradient descent.

4 Experiments

We design our experiments to investigate the following three questions: 1) How well does our method perform on different multi-step manipulation tasks? 2) How important does it perform in various complexities of the same task? 3) What kind of robot behaviors does our method produce?

4.1 Experimental Setup

Environment. We construct a simulated platform to evaluate multi-step manipulation tasks using a real-time physics simulator [17]. The workspace setup includes a 7-Dof Sawyer robot arm, a table surface, and an overhead depth sensor (Kinect2) as in Fig. 1. A random set of rigid objects (ranging from 1 to 5) are drawn from a subset of the YCB Dataset [18] and placed on the table. The Sawyer robot holds a short stick as the tool to interact with the objects to complete a specified task goal.

The observation consists of segmented point cloud represented by $m \times n \times 3$ Cartesian coordinates in the 3D space, where m is the number of movable objects and $n = 256$ is the fixed number of points we sampled on each object. The state is composed of the $m \times 3$ object centers and $m \times 64$ geometric features processed from the segmented point cloud, where the geometric features are extracted using a pretrained PointNet [19]. The robot performs planar pushing actions by the stick using position control. Each push is a straight line motion with limited moving distance of 0.1 meters along x- and y-axes. The action of each step is defined as a tuple of coordinates which represents the initial and delta-positions of the robot end-effector. The planning horizon is $T = 30$ steps and each episode terminates after 60 steps. Every $H = 3$ steps, the robot arm moves out of the camera view to take an unoccluded image from the environment for replanning.

Tasks. We design three multi-step manipulation tasks, listed in ascending order of complexity: *Clearing*, *Insertion*, and *Crossing*. A reward function is given to the robot for achieving the task goal. All tasks share the same MDP formulation in Sec. 2, while the arrangement of the scene and the reward functions are constructed differently in each task. None of the tasks are seen by the model during training time. The three tasks are illustrated in Fig. 2 with additional details in the Appendix.

In each task, dense and sparse reward functions are defined respectively. The dense reward function moderates the task complexity by providing intermediate reward signals, such as the Euclidean distance to the goal. While the sparse reward function only returns a positive value at the end of task completion, which poses a more intense challenge to plan ahead in a large search space requiring strategic interaction with different objects in diverse yet meaningful manner.
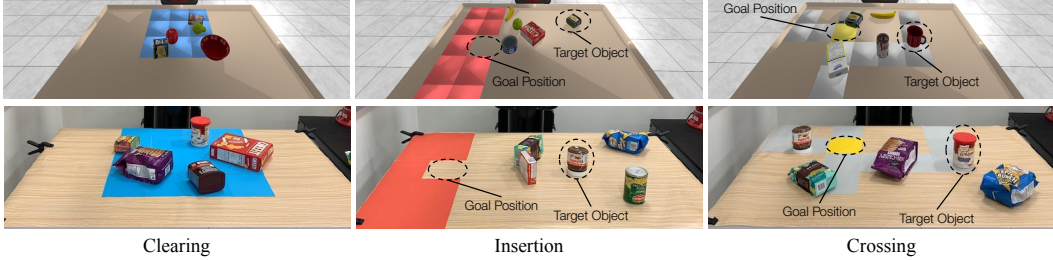
Figure 2: The environments of the three tabletop multi-step manipulation tasks in simulation and the real world. Objects and their initial placements are randomized in each episode.(see details in Sec. 4.1).

Baselines. We compare our method with a set of baselines, all of which use learned dynamics model coupled with model-based planning by sampling actions from the original action space or a latent space. These include **MPC** which samples from original state and action spaces, **CVAE-MPC** [10] which learns a flat generative model to sample actions, and **SeCTAR** [11] which learns a latent space model for states and actions for planning.

Training. To train all methods, we collect $500,000$ random transitions of planar pushing using a random policy. The dataset is partitioned for training (90%) and validation (10%) for hyperparameter tuning. Training terminates after one million steps when no further decrease in the total loss is observed on the validation set.

4.2 Quantitative Comparisons

Our method and baselines are compared across the three different multi-step manipulation tasks. We evaluate each method in each experiment below with $1,000$ episodes given the same random seed. Each method draws $1,024$ samples in every planning step. We analyze the task success rate of each method in the three tasks by providing dense or sparse rewards and varying the scene complexity.

Task performance given dense and sparse rewards. We compare all methods with dense and sparse rewards as shown in Fig. 3(top row). The number is evaluated with 3 objects initialized in each environment: one target object and two obstacles in `Insertion` and `Crossing`. Across all tasks, our method outperforms all other model-based baselines under both dense and sparse rewards. Given the dense reward functions, it has considerable margins compared to the second best methods (10.0% in `Clearing`, 11.7% in `Insertion`, 13.9% in `Crossing`). Especially for the latter two tasks which requires longer-term planning, the performance gap is telling. The planning becomes harder under sparse rewards, where a naïve or greedy algorithm cannot easily find the good strategy. In this case, performances of most methods drops under sparse rewards. However, our method suffers less than 5% drop across all 3 tasks. The margins over the second best are 20.1% in `Clearing`, 14.6% in `Insertion`, 11.1% in `Crossing`. This result demonstrates a strong advantage of the multi-scale hierarchical-dynamics model of our method for long-term planning under sparse rewards.

Compared to the baselines, we note that our method efficiently rules out trajectories of unsuccessful effects in the high-level planning process and effectively finds the reachable sequence of states that will lead to the task goal. In the low-level planning, our method focuses on sampling actions that lead to such effects. We also observe that different baseline fail for different reasons. Intuitively, in a large search space, uniformly sampled actions are largely ineffective or infeasible. Therefore, only a small fraction of samples in MPC actually lead to plausible solutions, while others are fruitless. CVAE-MPC uses the action generator in a flat planning framework. It produces task performance comparable to our method with dense rewards which provide intermediate guidance for correction. But under the sparse rewards, its performance is significantly undermined due to increasing difficulty of finding a long-horizon plan. SeCTAR effectively eliminates the undesired sequence of states using the meta-dynamics model similar to our method. However, using an entangled latent space for generating both states and actions leads to poor quality of generated action samples. While out method avoids such problem by introducing an additional latent variable as the motion code. Empirically, we also found the training of SeCTAR requires a careful balance between the reconstruction losses of state and action in order to yield reasonable results, which is a challenge in itself.

Task performance against number of objects. Given sparse rewards, we vary the number of objects in the workspace to evaluate the model performance under different task complexity as shown in Fig. 3. More objects lead to a exponentially growing search space of feasible plans, due to the combinatorial nature of subtask dependencies. We see that increasing the number of objects leads to
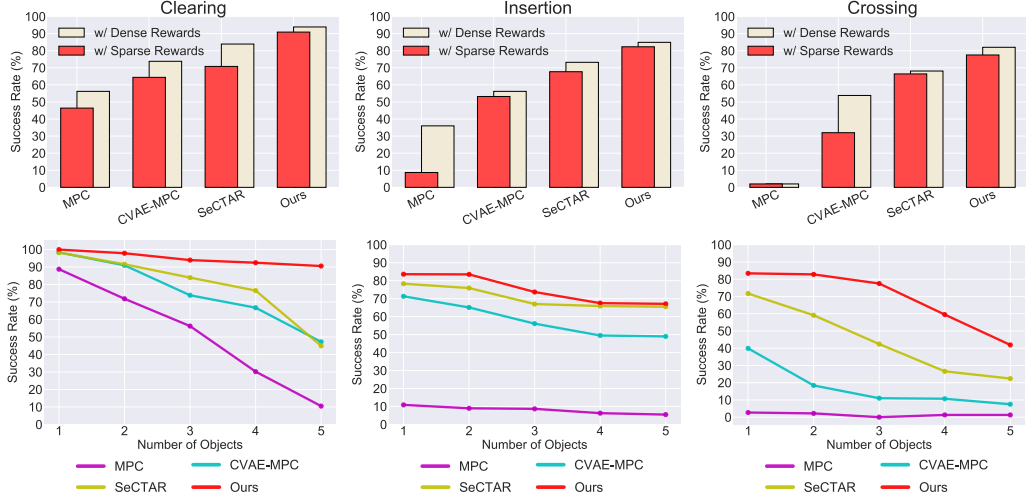
Figure 3: **Top Row:** Task performance with 3 objects given Dense and Sparse Rewards. In both cases, our method consistently outperforms all baseline methods. Compared to planning methods without hierarchies, our method is more robust under sparse rewards. **Bottom Row:** Task performance against number of objects. We evaluate the models' scalability and robustness with growing number of objects. We demonstrate that our method suffers the least from the increased complexity due to the cascaded generation of actions.

performance drop in all methods. However, our method shows the highest robustness and maintains the best performance. Especially in `Clearing`, the performance of our method only drops by 9.4% when increasing the number of objects from 1 to 5. While the performances of baseline methods drops 51.1%, 53.3% and 70.2% respectively.

Real-world experiments. We also evaluate our method in the real-world environments for the three tasks. The environment setup and the reward functions are identical to the simulated experiments. Since the observations of these tasks are based on point clouds which has relatively little reality gap, we can directly apply the model trained in the simulation in the real world without explicit adaptation. In each episode, we randomly initialize the environment with real-world objects including packaged foods, metal cans, boxes and containers. Among 15 evaluated episodes, out method achieves success rates of 93.3% in `Clearing`, 73.3% in `Insertion`, and 80.0% `Crossing`, which is comparable to our simulated experiments.

4.3   Qualitative Results

In Fig. 4, we visualize the planned trajectories predicted by our method. Example episodes of the `Crossing` task are shown in each row. In both simulation and the real world, our method effectively plans for subgoals towards achieving the task goal and executes the actions under the semantic and physical constraints. The planned actions successfully address different placements and types of objects in each episode.

5   Related Work

Hierarchical Planning. Learning diverse skills for hierarchical policies and controller is a common practice in control and robotics. One way to learn a set of different skills is to pre-speficy a set of modes and learn a policy for each mode. Each policy has a different parameterization and trained for a specific purpose. [20] composes subpolicies of different action modes using policy sketches provided beforehand and trains the composed policies as modular neural networks [21]. In robot manipulation, several methods define separate grasping and manipulation policies for pushing/tool use [22, 23, 24]. In contrast, our approach does not assume the action modes are specified beforehand. And our training procedure uses self-supervised interactions to learn a hierarchy of latent effects and actions, instead of relying on pre-specified low-level policy primitives.

Skill Learning. Another line of work aims to discover action modes from the data in an unsupervised manner. [25] defines skills as subpolicies that apply to different subsets of states, which can be learned by minimizing the Q-learning objective. Several multi-modal imitation learning algorithms propose
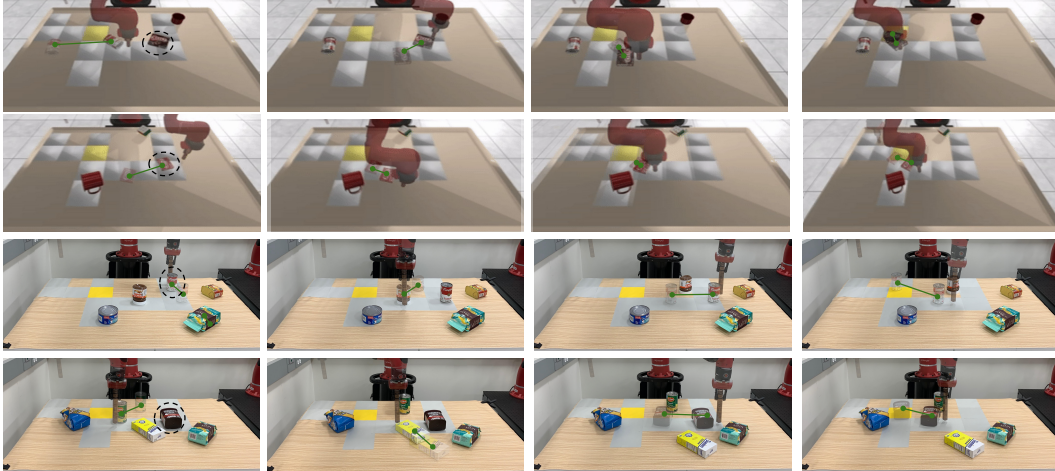
Figure 4: Visualization of the task execution in simulation and the real world using our trained model. Each row shows a different episode of the `Crossing` task. The executed subgoals are listed in temporal order from left to right. The subgoal is overlaid on the current state at each step and connected by a green line. The target object is marked by dashed circles at the first step.

to learn diverse behaviors from demonstrations [26, 27, 28]. These works assume the demonstration data is generated by multiple expert policies, which can be imitated separately in an adversarial learning framework. Our approach shares a similar goal as we aim to discover the action modes from data. While these works factorize action modes according to trajectories or subsets of states, our approach learns modes of actions by *how they affect* the environment state.

Generative Models for Planning. In addition to the baselines discussed in Sec. 4, Visual MPC [7], CVAE-SBMP [10], and SeCTAR [11], several other methods can be used for dynamics model learning. [10] uses conditional variational autoencoder (CVAE) [14, 29] to learn a generative model to draw collision-free samples from the action space. This idea is further extended for collision-free motion planning [30]. Co-Reyes et al. [11] use a policy decoder to generate sequence of actions and a state decoder to predict the dynamics given the same latent feature as the input. The two decoders are jointly trained to encourage the predicted future states are consistent with the resultant states caused by the generated actions. Both decoders receive the same latent feature as input.

The main notable difference is that most of these methods represent the probability distribution of a single action mode, where the data is often deliberate for the task. While, the hierarchical dynamics model in CAVIN decouples the model learning into latent code for effects and motion codes, each of which can guide the action sampling. And finally the consistency action sampling is ensured through dynamics prediction over a self-supervised dataset.

6 Conclusion

In this work, we proposed a hierarchical planning framework using learned latent spaces for multi-step manipulation tasks. Our model improves the efficiency of planning in complex tasks with high-dimensional continuous state and action spaces by exploiting the hierarchical abstraction of action spaces. Our model hierarchically performs model-based planning at two different temporal resolutions: the high-level prediction of effects and the low-level predictions of motions. We can thus perform hierarchical planning, where action sampling is done by first predicting a sequence of desired subgoals towards a task goal and second generating a sequence of actions to realize these effects. We evaluate our method in long-horizon robotic manipulation tasks with raw sensory inputs. The experiment results indicate that our method outperforms strong model-based baselines by large margins, as a result of its efficient sampling of effective generative process of actions.

References

[1] J. H. Wimpenny, A. A. Weir, L. Clayton, C. Rutz, and A. Kacelnik. Cognitive processes associated with sequential tool use in new caledonian crows. *PLoS One*, 4(8):e6471, 2009.

[2] W. Kohler. *The mentality of apes*. Routledge, 2018.

[3] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. *IEEE International Conference on Robotics and Automation*, pages 1470–1477, 2010.

[4] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. J. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 639–646, 2014.

[5] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in neural information processing systems*, pages 3338–3346, 2014.

[6] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. *Advances in Neural Information Processing Systems*, 2016.

[7] C. Finn and S. Levine. Deep visual foresight for planning robot motion. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793, 2017.

[8] R. Y. Rubinstein and D. P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. Springer Science & Business Media, 2013.

[9] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2146–2153. IEEE, 2017.

[10] B. Ichter, J. Harrison, and M. Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094. IEEE, 2018.

[11] J. D. Co-Reyes, Y. Liu, A. Gupta, B. Eysenbach, P. Abbeel, and S. Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. *arXiv preprint arXiv:1806.02813*, 2018.

[12] Y. Chandak, G. Theocharous, J. Kostas, S. Jordan, and P. S. Thomas. Learning action representations for reinforcement learning. *arXiv preprint arXiv:1902.00183*, 2019.

[13] T. Kurutach, A. Tamar, G. Yang, S. J. Russell, and P. Abbeel. Learning plannable representations with causal infogan. In *Advances in Neural Information Processing Systems*, pages 8733–8744, 2018.

[14] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[15] E. Camacho and C. Bordons. *Model Predictive Control*. Springer, 2007.

[16] A. Santoro, D. Raposo, D. G. T. Barrett, M. Malinowski, R. Pascanu, P. W. Battaglia, and T. P. Lillicrap. A simple neural network module for relational reasoning. In *NIPS*, 2017.

[17] E. Coumans and Y. Bai. pybullet, a python module for physics simulation, games, robotics and machine learning. http://pybullet.org/, 2016.

[18] B. Çalli, A. Singh, A. Walsman, S. S. Srinivasa, P. Abbeel, and A. M. Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. *2015 International Conference on Advanced Robotics (ICAR)*, pages 510–517, 2015.

[19] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[20] J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 166–175. JMLR. org, 2017.

[21] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48, 2016.

[22] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. IEEE, 2018.

[23] K. Fang, Y. Zhu, A. Garg, A. Kuryenkov, V. Mehta, L. Fei-Fei, and S. Savarese. Learning task-oriented grasping for tool manipulation from simulated self-supervision. *Robotics: Science and Systems (RSS)*, 2018.

[24] M. Danielczuk, A. Kurenkov, A. Balakrishna, M. Matl, D. Wang, R. Martín-Martín, A. Garg, S. Savarese, and K. Goldberg. Mechanical search: Multi-step retrieval of a target object occluded by clutter. *arXiv preprint arXiv:1903.01588*, 2019.

[25] S. Thrun and A. Schwartz. Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems*, 1994.

[26] Y. Li, J. Song, and S. Ermon. Infogail: Interpretable imitation learning from visual demonstrations. In *Advances in Neural Information Processing Systems*, pages 3812–3822, 2017.

[27] K. Hausman, Y. Chebotar, S. Schaal, G. Sukhatme, and J. J. Lim. Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 1235–1245, 2017.

[28] A. Tamar, K. Rohanimanesh, Y. Chow, C. Vigorito, B. Goodrich, M. Kahane, and D. Pridmore. Imitation learning from visual data with multiple intentions. In *International Conference on Learning Representations*, 2018.

[29] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pages 3483–3491, 2015.

[30] B. Ichter and M. Pavone. Robot motion planning in learned latent spaces. *IEEE Robotics and Automation Letters*, 2019.

A Task Descriptions

We use the following three multi-stage tabletop manipulation tasks to compare our method with baselines. These tasks involve multi-object interactions with complex dynamics, which requires the robot to perform long-horizon reasoning to devise an effective sequence of actions to complete each task from visual observations. The three tasks are illustrated in Fig. 2.

`Clearing`: The goal is to clear a region (in blue masks) on the table by moving away all objects within. Success is achieved when none of the objects remains in the workspace.

`Insertion`: A slot is placed randomly along one of the sides of the table, surrounded by restricted area indicated in red masks. A target object is specified while other movable objects are treated as distracting obstacles. The success is achieved when the target object is moved into the slot. If any object moves into the restricted area the task fails.

`Crossing`: A "bridge" is randomly constructed (in grey masks) on the table and the target object is placed on a random starting position on the bridge. The robot needs to move the target object to a goal position (in golden masks) on the other side of the bridge and moves away obstacles on the way. If the target object leaves the bridge the task fails.

In addition to the task constraints mentioned above, we also terminates an episode when objects leave the workspace of the robot. The model receives a reward of 100 when reaching the goal in each of the task and a penalty of -100 when the episode terminates because of violation of the task constraints. For dense rewards, we compute the distance that the target object moves towards the goal position in terms of meters. For clearing, we set virtual goals at the edge of the tables to encourage the robot to move objects out of the masked region.

B Baselines

Here we describe how we adapt each baseline method to the multi-step manipulation tasks in this paper and compare with our model design in the context of this paper.

MPC runs sampled-based planning by directly drawing samples from the original action space without a learned action generator.

CVAE-MPC adapts prior work [10] to learn a conditional VAE to sample collision-free trajectories for motion planning without a high-level dynamics model. We use their model to sample sequences of actions, same with our action generator, to compare with our model.

SeCTAR [11] uses a VAE for jointly sampling trajectories states and actions from the latent space in a self-consistent manner. In SeCTAR, both state and actions are decoded from a single latent variable (which can be considered as the effect code c in our case). In this way, the generative process of the states is equivalent to our high-level dynamics model, while the generative process of actions can be considered as a action generator given only c but not z. To enable replanning every T steps, we use the same action generator as in this paper to generate action sequences of T steps at once instead recursively predicting for each time step in [11].

C Implrmentation Details

We implement the modules using relational networks [16] consisting of 64 units per layer. All model variants and baselines use the same network architecture for their module counterparts. We use 32-dimensional latent codes for CVAE-MPC and SeCTAR and 16-dimensional c and z (totally 32-dimensional) to have comparisons using latent features of equivalent dimensions.

We use L2 losses for reconstructing actions and states in the ELBO of cascaded variational inference. For states, we sum up the L2 losses of the positions and geometric features of all objects.