TAPS: Task-Agnostic Policy Sequencing

Christopher Agia*, Toki Migimatsu*, Jiajun Wu, Jeannette Bohg

Abstract—Advances in robotic skill acquisition have made it possible to build general-purpose libraries of primitive skills for downstream manipulation tasks. However, naively executing these learned primitives one after the other is unlikely to succeed without accounting for dependencies between actions prevalent in long-horizon plans. We present Task-Agnostic Policy Sequencing (TAPS), a scalable framework for training manipulation primitives and coordinating their geometric dependencies at plan-time to efficiently solve long-horizon tasks never seen by any primitive during training. Based on the notion that Q-functions encode a measure of action feasibility, we formulate motion planning as a maximization problem over the expected success of each individual primitive in the plan, which we estimate by the product of their Q-values. Our experiments indicate that this objective function approximates ground-truth plan feasibility and, when used as a planning objective, reduces myopic behavior and thereby promotes task success. We further demonstrate how TAPS can be used for task and motion planning by estimating the geometric feasibility of candidate action sequences provided by a task planner. We evaluate our approach in simulation and on a real robot.

I. INTRODUCTION

Performing sequential manipulation tasks requires a robot to reason about dependencies between actions. Consider the example in Fig. 1, where the robot needs to grab an object outside its workspace by first using an L-shaped hook to pull the target object closer. How the robot picks up the hook affects whether the target object will be reachable.

Traditionally, planning actions to ensure the geometric feasibility of a sequential manipulation task is handled by motion planning [1–3] which typically requires privileged knowledge about the environment state and its dynamics. Learning-based approaches [4–6] can acquire primitive skills without requiring this privileged information. However, using learned primitives to perform sequential manipulation tasks is an unsolved problem. While myopically executing learned primitives one after another may solve a small subset of sequential manipulation tasks, solving more complex tasks requires planning with these primitives to ensure the geometric feasibility of the entire sequence.

Prior work focuses on sequencing learned primitives at train time to solve a single [7] or a small set of long-horizon tasks [8]. Such methods are limited when facing arbitrary long-horizon tasks, since they need to be trained on all the possible long-horizon tasks or primitive sequences they might encounter at test time. In our framework, we assume that a task planner provides us with a sequence of abstract primitives at test time that will then be grounded with concrete actions by sequencing learned primitives. This makes our method task-agnostic, since we can sequence primitives to solve long-horizon tasks not seen at train time.

The key insight of our method, Task-Agnostic Policy Sequencing (TAPS), is that geometric preconditions of learned primitives are implicitly encoded by Q-functions that model the expected success of the primitive given the current state and action. We use off-the-shelf RL algorithms to acquire Q-functions, and then define a planning objective to maximize all the Q-functions in the primitive sequence to ensure their feasibility. Evaluating downstream Q-functions on future states requires learning a dynamics model that can forward predict future states. We also use uncertainty quantification (UQ) to avoid visiting states and planning actions that are out-of-distribution (OOD) for the learned primitives. We train all of these components independently per primitive, making it easy to gradually expand a library of primitives without the need to retrain existing ones.

Our contributions are three-fold: we propose 1) a framework to train an extensible library of task-agnostic primitives, 2) a planning method to sequence primitives towards arbitrary long-horizon tasks, and 3) a method to solve Task and Motion Planning (TAMP) problems with learned primitives. In extensive experiments, we demonstrate that TAPS outperforms baselines for solving long-horizon tasks with complex geometric dependencies between actions. We also demonstrate that our framework works on a real robot.

II. RELATED WORK

A. Primitive skill learning

How to represent and acquire composable primitive skills is a widely studied problem in robotics. A broad class
of methods uses Learning from Demonstration (LfD) [5].
Dynamic Movement Primitives (DMPs) [9–11] are a form of
LfD that learns the parameters of dynamical systems encoding
movements [12–15]. More recent extensions integrate
DMPs with deep neural networks to learn more flexible
policies [16, 17]; for instance, to acquire a large library of
skill policies from human video demonstrations [18]. Skill
discovery methods instead identify action patterns in offline
datasets [19] and either distill them into policies [20, 21]
or extract skill priors for use in downstream tasks [22, 23].
Primitive skills can also be acquired via trial-and-error with
Reinforcement Learning (RL) [24–28] and offline RL [29].

An advantage of our planning framework is that it is
agnostic to the types of primitives employed, requiring
only that it is possible to predict the probability of the
skill’s success given the current state and skill parameter
(action). Here, we learn policies over parameterized action
primitives [30] which can express a range of manipulation
skills with a handful of skill parameters, empowering the
search for feasible parameter values with motion planning.

B. Long-horizon robot planning

Once primitive skills have been acquired, using them
to perform sequential manipulation tasks remains an open
challenge. [31, 32] propose data-driven methods to determine the
symbolic feasibility of primitives and only control their
timing, while we seek to ensure the geometric feasibility of
primitives by controlling their trajectories. Other techniques
rely on task planning [33, 34], subgoal planning [35], or
meta-adaptation [36, 37] to guide the sequencing of learned
primitives to novel long-horizon goals. However, the tasks
considered in these works do not feature rich geometric de-
pendencies between actions that necessitate motion planning.

The options framework [38] and parameterized action
MDPs [39] outline control paradigms that train a high-
level policy to engage low-level policies [40, 41] or
primitives [7, 42, 43] towards a goal. [43] proposes a
hierarchical RL method that uses the value functions of
lower-level policies as the state space for a higher-level
RL policy. Our work is also related to model-based RL
methods which jointly learn dynamics and reward models to
guide planning [45–47], policy search [48, 49], or combine
both [50, 51]. While these methods demonstrate that policy
hierarchies and model-based planning can enable RL to solve
long-horizon problems, they are typically trained in the con-
text of a single task. In contrast, we seek to plan with lower-
level primitives policies to solve tasks never seen before.

Closest in spirit to our work is that of Xu et al. [8], Deep
Affordance Foresight (DAF), which proposes to learn a dy-
namics model, skill-ancient affordances (i.e. value functions),
and a skill proposal network that serves as a higher-level
RL policy. We identify several drawbacks with DAF: first,
because DAF relies on multi-task experience for training,
generalizing to tasks outside the distribution of training
tasks may be difficult; second, the dynamics, affordance
models, and skill proposal need to be trained synchronously,
making scalability an issue when training a large library of
primitives; third, their planner samples actions from uniform
random distributions, which prevents it from scaling to high-
dimensional action spaces and long horizons. We are unique
in that our dynamics, skill policies, and affordances (Q-
factors) are learned independently per primitive, and then
without additional training, we combine them at planning
time to solve arbitrary long-horizon tasks. We compare our
method against DAF in the planning experiments.

C. Task and motion planning

TAMP solves problems that require a combination of sym-
bolic and geometric reasoning [2, 52]. DAF learns a skill pro-
posal network to replace the typical task planner in TAMP,
akin to [53]. Another prominent line of research learns com-
ponents of the TAMP system, often from a dataset of precom-
puted solutions [54–58]. The problems considered in our pa-
per involve complex geometric dependencies between actions
that are typical in TAMP. However, TAPS only performs the
geometric reasoning part and by itself is not a TAMP method.
We demonstrate in experiments that it can be effectively
combined with symbolic planners to solve TAMP problems.

III. PROBLEM SETUP

Our goal is to solve long-horizon manipulation tasks that
require sequential execution of primitive policies from a library
\( L = \{\pi^k \}, \) Each primitive \( k \) is associated
with a contextual bandit, or a single timestep Markov
Decision Process (MDP)

\[
\mathcal{M}^k = (\mathcal{S}^k, \mathcal{A}^k, T^k, R^k, \rho^k),
\]

where \( \mathcal{S}^k \) is the state space, \( \mathcal{A}^k \) is the action space,
\( T^k(s^k \mid s^k, a^k) \) is the transition model, \( R^k(s^k, a^k) \)
is the binary reward function, and \( \rho^k(s^k) \) is the initial state
distribution.

A long-horizon domain is one in which each timestep
involves the execution of one primitive. Upon executing
a primitive action \( a^k \), the state evolves according to
the primitive transition dynamics \( T^k(s^k \mid s^k, a^k) \). A long-
horizon domain is specified by the tuple

\[
\mathcal{M} = (\mathcal{M}^{1:k}, \overline{\mathcal{S}}, \overline{T}^{1:k}, \overline{\mathcal{P}}^{1:k}, \overline{\Psi}^{1:k}),
\]

where \( \mathcal{M}^{1:k} \) is the set of MDPs whose primitives can be
executed in the long-horizon domain, \( \overline{\mathcal{S}} \) is the state space
of the long-horizon domain, \( \overline{T}^{k}(\overline{s}^k \mid \overline{s}^k, \overline{a}^k) \) is an extension
of primitive dynamics \( T^k(s^k \mid s^k, a^k) \) that models how the
entire long-horizon state evolves with action \( a^k \), \( \overline{\mathcal{P}}^{k}(\overline{\pi}^k) \)
is an extension of primitive initial state distributions \( \rho^k(s^k) \)
over the long-horizon state space, and \( \overline{\Psi}^k : \overline{\mathcal{S}} \rightarrow \overline{\mathcal{S}}^k \)
is a function that maps from the long-horizon state space to the
state space of primitive \( k \). We assume that the dynamics
\( T^k(s^k \mid s^k, a^k), \overline{T}^{k}(\overline{s}^k \mid \overline{s}^k, \overline{a}^k) \) and initial state distributions
\( \rho^k(s^k), \overline{\mathcal{P}}^{k}(\overline{\pi}^k) \) are unknown for all primitives \( k \).

Note that while the primitives may have different state
spaces \( \mathcal{S}^k \), they must all be obtainable from the long-horizon
state space \( \overline{\mathcal{S}} \) via \( \overline{\Psi}^k \). This is to ensure that the primitives
can be used together in the same environment to perform
the objective can be cast as the expectation
long-horizon task is considered successful if every primitive
is framed as the optimization problem
ground the plan skeleton
M
-Ask
-Plan

A. Grounding primitive sequences with action plans

We assume we are given a plan skeleton of primitives
π
= [π
1
,..., π
H
] ∈ \mathcal{L}^H
should be executed sequentially
to solve a long-horizon task. Let \mathcal{M}_h
with subscript h denote
the primitive MDP corresponding to the h-th primitive
in the sequence (in contrast to \mathcal{M}^k
with superscript k, which
denotes the k-th primitive MDP in the primitive library).
A long-horizon task is considered successful if every
primitive reward r
1
,..., r
H
reached during execution is 1.

Given an initial state \pi \in \mathcal{S}
our problem is to
ground the plan skeleton \pi
with an action plan
\xi = [a
1
,..., a
H
] ∈ \mathcal{A}_1 × \cdots × \mathcal{A}_H
that maximizes the probability of succeeding at the long-horizon task.
This is framed as the optimization problem \arg\max_{a
1
,..., a
H
} J,
where the maximization objective J is the task success probability
J(a
1
,..., a
H
; \pi
1
) = p(r
1
= 1,..., r
H
= 1 \mid \pi
1
, a
1
,..., a
H
).

r
1
,..., r
H
are the primitive rewards received at each timestep.

By the Markov assumption, rewards are conditionally independent
given states and actions. We can express the probability
of task success as the product of reward probabilities
J(a
1
,..., a
H
; \pi
1
) = \Pi_{h=1}^{H} p(r_h = 1 \mid \pi_h, a_h).

With the long-horizon dynamics model \mathcal{T}^k(\mathcal{S} \mid \pi, a^k),
the objective can be cast as the expectation
J = \mathbb{E}_{\pi \sim \pi
1
,..., \pi
H
 \sim \mathcal{T}^k \mid \pi
1
,..., \pi
H
} [\Pi_{h=1}^{H} p(r_h = 1 \mid \pi_h, a_h)].

Because the primitive rewards are binary, the primitive
success probabilities are equivalent to Q-values:
\begin{align*}
p(r_h \mid \pi_h, a_h) &= \mathbb{E}[r_h \mid \pi_h, a_h] = Q_h(\Psi_h(\pi_h), a_h).
\end{align*}

The objective is finally expressed in terms of Q-values:
J = \mathbb{E}_{\pi \sim \pi
1
,..., \pi
H
 \sim \mathcal{T}^k \mid \pi
1
,..., \pi
H
} [\Pi_{h=1}^{H} Q_h(\Psi_h(\pi_h), a_h)].

This planning objective is simply the product of Q-values
evaluated along the trajectory (\pi
1
, a
1
,..., \pi
H
, a
H
),
where the states are predicted by the long-horizon dynamics
model: \pi
2
= \mathcal{T}_1(\pi
1
, a
1
),..., \pi
H
= \mathcal{T}_{H-1}(\pi_{H-1}, a_{H-1}).

Optimizing this objective requires access to Q-functions
Q^k(s^k, a^k) and dynamics models \mathcal{T}^k(\pi, a^k).
In Sec. V, we propose a training procedure for learning these models.

B. Ensuring action plan feasibility

A plan skeleton \pi = [\pi
1
,..., \pi
H
] is feasible only if, for every
pair of consecutive primitives i and j, there is a non-zero overlap
between the terminal state distribution of i and
the initial state distribution of j. More formally,
\[
\mathbb{E}_{\pi \sim \pi
i\sim\pi
j \sim \mathcal{T}^k \mid \pi
i\sim\pi
j} [\mathcal{T}_1(\pi
j \mid \pi
i, a_i)] > 0,
\]
where \pi
i
and \pi
j
are the initial state distributions for
primitives i and j, respectively, and a
i
is uniformly distributed with respect to action space \mathcal{A}_i
for primitive i. Given a state \pi
i
\sim \pi
j
, it is part of the planner’s job
to determine an action a
i
that produces a valid subsequent
state \pi
j
\sim \pi
j
if one exists. Failing to do so constitutes
an OOD event for primitive j where the subsequent state
\pi
j
has drifted beyond the region of the state space
where Q_j(\Psi_j(\pi_j), a_j) is well-defined and \pi
j
is executable.

Neglecting state distributional shift over an action plan
\xi may degrade the quality of objective function J with
spuriously high Q-values. Moreover, Eq. 8 cannot be
explicitly computed to determine the validity of actions
because the initial state distributions of all primitives
\pi
j
are unknown. We can detect OOD states (and actions)
by performing uncertainty quantification (UQ) on
the Q-functions Q^k(s^k, a^k).
Filtering out Q-values with high uncertainty
would result in action plans \xi that are robust
have low uncertainty) while maximizing the task feasibility
objective. We discuss efficient methods for training UQ
models on learned Q-functions in Sec. V-C.

V. TRAINING PRIMITIVES

A. Policies

One of the key advantages of our approach is that
the primitive policies can be trained individually,
and then composed together at test time to solve
unseen sequential tasks. For each primitive, we want to obtain a policy
\pi^k : \mathcal{S} \rightarrow \mathcal{A}^k
that can solve the primitive-specific task
specified by its MDP \mathcal{M}^k.
In addition to the policy, we also want to learn a Q-function for each primitive:
\[
Q^k(s^k, a^k) = \mathbb{E}_{s^k \sim \mathcal{T}^k \mid s^k, a^k} [R^k(s^k, a^k)].
\]

Our framework is agnostic to the method for acquiring
the policy and Q-function. Many deep RL algorithms are able to learn
the policy (actor) and Q-function (critic) simultaneously with
unknown dynamics [59, 60]. We therefore leverage
off-the-shelf RL algorithms to learn a

One might consider maximizing the \textit{sum} of Q-values instead of
the product, but this may not reflect the probability of task success.
For example, if we want to optimize a sequence of ten primitives, consider a
plan that results in nine Q-values of 1 and one Q-value of 0, for a total
sum of 9. One Q-value of 0 would indicate just one primitive failure,
but this is enough to cause a failure for the entire task. Compare this to a plan
with ten Q-values of 0.9. This plan has an equivalent sum of 9, but it is
preferable because it has a non-zero probability of succeeding.
B. Dynamics

The dynamics models are used to predict future states at which each downstream Q-function in the plan will be evaluated. We learn a deterministic model for each primitive $k$:

$$\bar{s}' = T^k(s, a^k).$$  \hspace{1cm} (10)

Using the forward prediction loss, each dynamics model is trained on the state transition experience $(\bar{s}, a^k, s')$ collected during policy training for primitive $k$:

$$L_{dynamics}(T^k; \bar{s}, a^k, s') = \|T^k(\bar{s}, a^k) - s'\|^2. \hspace{1cm} (11)$$

C. Uncertainty quantification

Measuring the epistemic uncertainty over the Q-values will allow us to identify when dynamics-predicted states and planned actions drift OOD for downstream primitive critics $Q^k$. We leverage recent advances in neural network UQ, taking an approximate Bayesian computational approach to obtain an explicit Gaussian posterior distribution over Q-values

$$p\left(Q^k \mid s^k, a^k, D^k, w^k\right) = \mathcal{N}\left(\mu_{Q^k}, \sigma_{Q^k}; w^k\right)$$  \hspace{1cm} (12)

with sketching curvature for OOD detection (SCOD) [62]. We apply SCOD to obtain the weights $w^k$ from which the posterior distributions over each critic $Q^j$ are derived. The advantages of SCOD over common UQ techniques [63, 64] are that it imposes no train-time dependencies on any algorithms used in our framework, and it requires only the experience $(\bar{s}, a^k) \sim D^k$ collected over the course of training primitive policies $\pi^k$.

VI. PLANNING WITH PRIMITIVES

A. Planning action sequences

To find action sequences that maximize the probability of long-horizon task success (Eq. 7), we use sampling-based methods: shooting and cross entropy method (CEM) [65]. In shooting, we simply sample action plans $\xi = a_{1:H} \in \mathcal{A}_1 \times \cdots \times \mathcal{A}_H$, and choose the one with the highest predicted objective score. CEM is an extension of shooting, where the action sampling distribution is iteratively refined to fit a fraction of the population with the highest objective scores. With small action spaces and short primitive sequences, randomly sampling action plans from a uniform distribution may often be sufficient. However, this strategy suffers from the curse of dimensionality and does not scale to the large action spaces and long primitive sequences that we consider. Meanwhile, directly sequencing actions $a^k \sim \pi^k(s^k)$ from the primitive policies learned via off-the-shelf RL methods (V-A) produces myopic plans that rarely succeed for tasks with complex geometric dependencies between actions.

The policies can be leveraged to initialize a sampling-based search with an action plan that is likely to be closer to an optimal plan than one sampled uniformly at random. We therefore use two variants of shooting and CEM, termed policy shooting and policy CEM, which sample actions from Gaussian distributions $a^k \sim \mathcal{N}(\pi^k(s^k), \sigma)$, where the mean is the action predicted by the policy and the standard deviation is a planning hyperparameter.

B. Task and motion planning

In Sec. IV-A, we start with the assumption that the plan skeleton $\tau = \pi_{1:H}$ is given, although our approach is agnostic to where it comes from. For example, the plan skeletons can be computed by symbolic planners in a TAMP loop, invoking our planner to evaluate the geometric feasibility of proposed plans. We can describe TAMP domains by combining the Planning Domain Description Language (PDDL) [66] with the long-horizon domain $\mathcal{M}$ in Eq. 2:

$$\mathcal{D}_{TAMP} = (\Phi, \Pi^{1-K}, \mathcal{M}).$$  \hspace{1cm} (13)

$\Phi$ is the set of predicates that describe binary-valued symbolic properties of objects, and $\Pi^{1-K}$ are the symbolic actions, one per primitive, with symbolic pre-conditions and effects that describe how each primitive modifies the symbolic state of the world. A TAMP problem is specified by the tuple

$$\mathcal{P}_{TAMP} = (O, s_1, s_1),$$  \hspace{1cm} (14)
where $\mathcal{O}$ is the set of symbolic objects in the problem, $g$ is the symbolic goal specified as a first-order logic formula, $\mathbf{s}_1$ is the initial symbolic state represented as a set of propositions (predicates instantiated by object arguments), and $\mathbf{s}_1 \in \mathcal{S}$ is the initial state for $\mathcal{M}$.

Given a TAMP problem, the task planner will solve the PDDL problem $(\mathcal{O}, g, \mathbf{s}_1)$ to produce a candidate plan skeleton $\tau = \mathbf{a}_{1:H}$. We then perform motion planning as described in VI-A to find a feasible action plan $\xi = \mathbf{a}_{1:H}$ for the candidate plan skeleton. The task planner continues to find the next candidate plan skeleton. After some termination criterion is met, we return the candidate plan skeleton and grounded action plan with the highest probability of task success (Eq. 7).

### VII. Experiments

In our experiments, we test the following hypotheses:

**H1** Maximizing the product of learned Q-functions (Eq. 7) translates to maximizing long-horizon task success.

**H2** Primitives trained with our framework are able to generalize to unseen long-horizon tasks.

**H3** Our planning method can be combined with a task planner to solve TAMP problems.

We evaluate our method on a 3D manipulation domain with 4 primitives: Pick$(a,b)$: pick $a$ from $b$; Place$(a,b)$: place $a$ onto $b$; Pull$(a,\text{hook})$: pull $a$ into the robot’s workspace with a hook; and Push$(a,\text{hook})$: push $a$ with a hook.

The state space $\mathcal{S}$ is a sequence of low-dimensional object states that contains information such as 6D poses. The primitive state spaces $\mathcal{S}^k$ are constructed so that the first $m$ object states correspond to the $m$ arguments of the corresponding primitive. For example, a state for Pick$(\text{box}, \text{rack})$ will contain first the box’s state, then the rack’s state, followed by a random permutation of the remaining object states.

The primitive action spaces $\mathcal{A}^k$ in this domain are all 4D. For example, an action for Pick$(a,b)$ specifies the 3D grasp position of the end-effector relative to the target $a$ and orientation about the world $z$-axis.

Our evaluation is on 9 different long-horizon tasks (i.e. plan skeletons $\tau$) in this domain (Fig. 3). The tasks cover a range of symbolic and geometric complexity, with plan skeleton lengths ranging from 4 to 10 actions. Each task involves geometric dependencies between actions, which motivate the need for planning. For each task, we use 100 randomly generated instances (i.e. object configurations) for evaluations.

**A. Product of Q-functions approximates task success (H1)**

We evaluate H1 by comparing it to an Oracle baseline that runs forward simulations with policy shooting to find plans that achieve ground-truth task success. Our method, on the other hand, uses learned Q-functions and dynamics to predict task success as the product of Q-functions. We expect that planning with this objective will come close to matching the task success upper bound provided by Oracle.

We compare several planning methods: Policy Shooting and Policy CEM, which use the learned policies to bias the action sampling distributions (VI-A), as well as Random Shooting and Random CEM, which use uniform action priors. We also compare with Greedy, which does not plan but greedily executes the policies. The evaluation metrics are ground truth task success, sub-goal completion rate (what percentage of actions in a plan are successfully executed), and predicted task success computed from Eq. 7.

Due to the significant amount of time required to run forward simulations for Oracle, we limit the number of samples evaluated during planning to 1000 for all methods. This is not enough to succeed at the most complex tasks, so in our evaluation, we only use the easiest task from the Hook Reach and Constrained Packing domains.

The results from both tasks are averaged and presented in Fig. 4. Oracle achieves the highest success rate, as expected, but is not perfect likely because 1000 samples are not enough to solve the tasks. Policy CEM nearly matches Oracle’s success rate, which demonstrates that maximizing the product of Q-functions is a good proxy for maximizing task success. Policy CEM also exhibits a low success prediction error, which demonstrates that the learned Q-functions and dynamics generalize well to these unseen long-horizon tasks. Meanwhile, planning with these learned models runs 4 orders of magnitude faster than Oracle and does not require ground truth knowledge about the environment state or dynamics.

**Policy Shooting** performs slightly worse than Policy CEM, which indicates CEM’s strength in finding local maxima. Random CEM and Random Shooting perform quite poorly, indicating that the planning space is too large (16D for these tasks) for random sampling. Greedy performs strongly, perhaps indicating that these simple tasks from the two domains may be doable without planning.

**B. TAPS primitives generalize to long-horizon tasks (H2)**

In this experiment, we specifically test the ability of our framework to solve 9 long-horizon tasks with geometric dependencies between actions. We compare against DAF [8], a state-of-the-art method for learning to solve TAMP problems. While DAF employs a skill proposal network that serves the role of a task planner, task planning is outside the scope of this paper, so we leave out skill proposal from this evaluation and compare only against the skills trained with DAF (DAF-Skills), which are comprised of dynamics and affordance models. DAF’s planning objective is similar to ours, except they evaluate the product
of affordances rather than Q-functions. We give DAF-Skills the same plan skeleton $\tau$ that is given to our method. We also augment DAF’s shooting planner with CEM.

Like other model-based RL methods, DAF requires training on a set of long-horizon tasks that is representative of the evaluation task distribution. We therefore train one DAF-Skills model per task (9 total) and run our evaluation on the same task. We also test the ability of these models to generalize to the other two tasks within the same domain (DAF-Gen). For the simplest tasks, we expect DAF-Skills to perform at least as well as our method, if not better, since it is trained on the same task used for evaluation. For the longer horizon tasks, it may perform worse, since the probability of completing a task drops exponentially with the length of the task, which means DAF-Skills will naturally get fewer chances to explore later steps. We train DAF-Skills and our method for 48 hours each, and allow up to 100,000 samples per dimension for planning.

The results are presented in Fig. 5. Our method with Policy CEM achieves the highest success rate for most tasks. DAF’s performance is inconsistent; while it achieves competitive success rates for 3 out of 9 tasks, it gets relatively low success on the others. This indicates that training the model directly on the long-horizon task may not be the most effective strategy. Our method of training each policy on primitive-specific environments and then generalizing to long-horizon tasks via planning is efficient from a training perspective, since the same trained primitives can be used for all tasks, and also scales better to longer horizons.

C. TAPS can be integrated into a TAMP framework (H3)

In this experiment, we combine our framework with a PDDL task planner as described in VI-B and evaluate it on two TAMP problems. In Hook Reach, the robot needs to decide the best way to pick up a block, which may or may not be in its workspace. In Constrained Packing, the robot needs to place a fixed number of objects on the rack but is free to choose among any of the objects on the table. To mimic what the robot might find in an unstructured, real-world environment, some of these objects are distractor objects that are initialized in ways not seen by the primitives during training (e.g. the blocks can be stacked, placed behind the base, or tipped over). The task planner may end up selecting these distractor objects for placing on the rack, but since the policies have not been trained to handle these objects, their predicted success may be unreliable. UQ is particularly important for such scenarios, so we introduce SCOD Policy CEM, which filters out candidate plans with high uncertainty in the predicted task success.

The results are presented in Fig. 6. Policy CEM achieves 97% success for the Hook Reach TAMP problem. SCOD Policy CEM suffers a slight performance drop. However, for Constrained Packing, which contains OOD states, SCOD Policy CEM strongly outperforms the other methods. Exploring different ways to integrate UQ into our planning framework is a promising direction for future work.

D. Real world sequential manipulation

We demonstrate that policies trained with our framework can be used to perform sequential manipulation tasks in a real robot environment. We take RGB-D images from a Kinect v2 camera and use manually tuned color thresholds to segment the objects. With these segmentations, we estimate object poses using the depth image, which is then used to construct the initial environment state $\pi_1$. Qualitative results are provided in the supplementary video.

VIII. Conclusion

We present a framework for sequencing task-agnostic policies that have been trained independently. The key to generalization is planning actions that maximize the probability of long-horizon task success, which we model using the product of learned Q-values. This requires learning a dynamics model to predict future states and using UQ to filter out OOD states that the primitives do not support. The result is a library of primitives that can be composed to solve arbitrary long-horizon tasks with complex geometric dependencies between actions. Future work includes investigating methods to handle high-dimensional observations, combining the library of learned primitives with a set of handcrafted primitives, and exploring planning objectives that seek to optimize trajectories, beyond finding geometrically feasible trajectories.
K. Chua, R. Calandra, R. McAllister, and S. Levine, “Deep reinforcement learning in a handful of trials using probabilistic dynamics models,” Advances in neural information processing systems, vol. 31, 2018.

D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, “Learning latent dynamics for planning from pixels,” in International conference on machine learning. PMLR, 2019, pp. 2555–2565.

M. Janner, J. Fu, M. Zhang, and S. Levine, “When to trust your model: Model-based policy optimization,” Advances in Neural Information Processing Systems, vol. 32, 2019.

D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, “Dream to control: Learning behaviors by latent imagination,” arXiv preprint arXiv:1912.01603, 2019.

K. Xie, H. Bharadhwaj, D. Hafner, A. Garg, and F. Shkurti, “Skill transfer via partially amortized hierarchical planning,” 2020.

R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak, “Planning to explore via self-supervised world models,” in International Conference on Machine Learning. PMLR, 2020, pp. 8583–8592.

C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, “Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning,” in Proceedings of the International Conference on Automated Planning and Scheduling, vol. 30, 2020, pp. 440–448.

C. Wang, D. Xu, and L. Fei-Fei, “Generalizable task planning through representation pretraining,” arXiv preprint arXiv:2205.07993, 2022.

Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, “Active model learning and diverse action sampling for task and motion planning,” in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018, pp. 4107–4114.

D. Driess, J.-S. Ha, and M. Toussaint, “Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image,” arXiv preprint arXiv:2006.05398, 2020.

R. Chitnis, T. Silver, B. Kim, L. P. Kaelbling, and T. Lozano-Perez, “Camps: Learning context-specific abstractions for efficient planning in factored mdps,” arXiv preprint arXiv:2007.13202, 2020.

T. Silver, R. Chitnis, A. Curtis, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, “Planning with learned object importance in large problem instances using graph neural networks,” in Proceedings of the AAAI conference on artificial intelligence, vol. 35, no. 13, 2021, pp. 11 962–11 971.

D. Driess, J.-S. Ha, R. Tedrake, and M. Toussaint, “Learning geometric reasoning and control for long-horizon tasks from visual input,” in 2021 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2021, pp. 14 298–14 305.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” arXiv preprint arXiv:1509.02971, 2015.

S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in International conference on machine learning. PMLR, 2018, pp. 1587–1596.

T. Haarnoja, A. Zhou, K. Haritakainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel et al., “Soft actor-critic algorithms and applications,” arXiv preprint arXiv:1812.05905, 2018.

A. Sharma, N. Azizan, and M. Pavone, “Sketching curvature for efficient out-of-distribution detection for deep neural networks,” in Uncertainty in Artificial Intelligence. PMLR, 2021, pp. 1958–1967.

M. Ganaie, M. Hu et al., “Ensemble deep learning: A review,” arXiv preprint arXiv:2104.02395, 2021.

Y. Gal and Z. Ghahramani, “Dropout as a bayesian approximation: Representing model uncertainty in deep learning,” in International conference on machine learning. PMLR, 2016, pp. 1050–1059.

R. Rubinstein, “The cross-entropy method for combinatorial and continuous optimization,” Methodology and computing in applied probability, vol. 1, no. 2, pp. 127–190, 1999.

D. McDermott, M. Khailab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, PDDL—The Planning Domain Definition Language Version 1.2, 1998.