PLANNING TO PRACTICE: EFFICIENT ONLINE FINE-TUNING BY COMPOSING GOALS IN LATENT SPACE

Anonymous authors
Paper under double-blind review

ABSTRACT

General-purpose robots in real-world settings require diverse repertoires of behaviors to complete challenging tasks in unstructured environments. To address this problem, goal-conditioned reinforcement learning aims to train policies that can reach configurable goals for a wide range of tasks on command. However, such goal-conditioned policies are notoriously difficult and time-consuming to train from scratch. In this paper, we propose Planning to Practice (PTP), a method that makes it practical to train goal-conditioned policies for long-horizon tasks that require multiple distinct types of interactions to solve. Our approach is based on two key ideas. First, we decompose the goal-reaching problem hierarchically, with a high-level planner that sets intermediate subgoals using conditional subgoal generators in the latent space for a low-level model-free policy. Second, we propose a hybrid offline reinforcement learning approach with online fine-tuning, which uses previously collected data to pre-train both the conditional subgoal generator and the policy, and then fine-tune the policy via online exploration. This fine-tuning process is itself facilitated by the planned subgoals, which break down the original target task into short-horizon goal-reaching tasks that are significantly easier to learn. We conduct experiments in both the simulation and real world, in which the policy is pre-trained on demonstrations of short primitive behaviors and fine-tuned for temporally extended tasks that are unseen in the offline data. Our experimental results show that PTP can generate feasible sequences of subgoals that enable the policy to efficiently solve the target tasks.[1]

1 INTRODUCTION

Developing controllers that can solve a range of tasks and generalize broadly in real-world settings is a long-standing challenge in robotics. Such generalization in other domains such as computer vision and natural language processing has been attributed to training large machine learning models on massive datasets (Krizhevsky et al., 2012). Consequently, one promising approach to robustly handle a wide variety of potential situations that a robot might encounter is to train on a large dataset of robot behaviors. Prior work in robotics has demonstrated effectively generalization from training on large datasets for individual tasks, such as grasping (Levine et al., 2017; Yu et al., 2021). However, a general-purpose robot should be able to perform a wide range of skills, and should also be *taskable*. That is, it must be able to complete a specific task when specified by a human, including temporally extended tasks that require sequencing many skills together to complete. This topic has been studied in prior work on goal-conditioned reinforcement learning, where a robot aims to perform a task given a desired end state (Khazatsky et al., 2021; Chebotar et al., 2021; Kalashnikov et al., 2021). What remains to make these methods widely applicable for real-world robotics?

While goal-conditioned policies can be trained effectively for relatively short tasks, temporally extended multi-stage can pose a significant challenge for current methods. These tasks present a major exploration challenge during online learning, and a major challenge for credit assignment during offline learning.

In this paper, we aim to address these challenges by combining two ideas. The first is that long-horizon goal-reaching tasks can be decomposed into shorter-horizon tasks consisting of subgoals.

[1]Supplementary video: sites.google.com/view/planning-to-practice

The second is that these subgoals can be used to *fine-tune* a goal-conditioned policy online, even if its performance from offline data is poor.

The first idea enables us to address the exploration challenge, by automatically generating intermediate subgoals that can be "practiced" on the way to a longer-horizon final goal. In the framework of goal-conditioned RL, solving long horizon tasks can be reduced to the problem of optimization over a sequence of subgoals for the goal-conditioned policy, and this optimization over subgoals can be regarded as a kind of high-level planning, where the optimizer selects "waypoints" for achieving a distant goal. The high-level planner itself can use a learned high-level model.

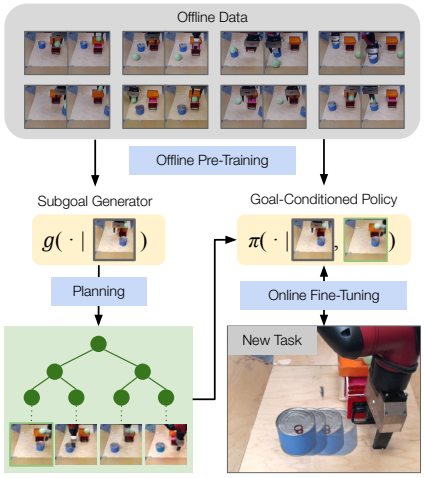

Figure 1: Our method, Plan to Practice (PTP), solves long-horizon goal-conditioned tasks by combining planning and fine-tuning. We begin with an offline dataset containing a variety of behaviors, and train a subgoal generator and goal-conditioned policy on this data. To learn a more complex multi-stage tasks, we optimize over subgoals using the subgoal generator, which corresponds to a planning procedure over (visual) subgoals, and fine-tune the policy with online RL by practicing these subgoals. This enables the robot to solve multi-stage tasks directly from images.

However, if we rely entirely on offline data, credit assignment challenges make it difficult to perform longer-horizon tasks even with subgoal planning. Even if the offline RL policy performs well on each individual skill, there may be errors from stitching skills together because the initial states of each stage diverge from the offline data when they are composed together. In practice, this leads to poor performance when using only offline training. Therefore, the second key idea in our work is to utilize subgoal planning not merely to *perform* a multi-stage task, but also to make it possible to *practice* that task to finetune it online. While online training for temporally extended tasks is ordinarily difficult, by addressing the exploration challenge with subgoal planning, we make it possible for the robot to practice a series of relatively short-horizon tasks, which makes this kind of finetuning feasible. Thus, the planner acts both as a higher level policy when performing the task, and as a scaffolding curriculum for finetuning the lower-level goal-conditioned policy By collecting data actively in a specific environment, we can directly experience the distribution shift and can use reinforcement learning to improve performance under this shift.

To this end, we propose Planning to Practice (PTP), an approach that efficiently trains a goal-conditioned policy to solve multi-step tasks by setting subgoals to exploit the compositional structure of the offline data. An outline is shown in Fig. 1. Our approach is based around a planner that composes generated subgoals to guide the goal-conditioned policy during an online fine-tuning phase. To propose diverse and reachable subgoals to form the candidate plans, we design a conditional subgoal generator based on conditional variational autoencoder (CVAE) (Sohn et al., 2015). Through training on the offline dataset, the conditional subgoal generator captures the distribution of reachable subgoals from a given state and generates sequences of subgoals from the learned latent space in a recursive manner. Our subgoal planning algorithm hierarchically searches for subgoals in a coarse-to-fine manner using multiple conditional subgoal generators that trained to generate goals at different temporal resolutions. Both the goal-conditioned policy and conditional subgoal generators are pre-trained on the offline data, and the policy is fine-tuned on the novel target task.

Our main contribution is a system for learning to solve long-horizon goal-reaching tasks by fine-tuning the goal-conditioned policy with subgoal planning in a learned latent space. We evaluate our approach on multi-stage robotic manipulation tasks with raw image observations and image goals in both simulation and the real world. After being pre-trained on short demonstrations of primitive interactions, our approach is able to find feasible subgoal sequences as plans for unseen final goals by recursively generating subgoals with the learned conditional subgoal generators. By comparing our approach with both model-free methods and prior approaches that optimize over subgoals, we

demonstrate that the produced plans significantly improve the learning efficiency and the resultant success rates during the online fine-tuning.

2 PROBLEM STATEMENT

In this paper, we consider the problem of learning to complete a long-horizon task specified by a goal image. The robot learns over a variety of initial configurations and goal distributions, which cover a range of behaviors such as opening or closing a drawer, and picking, placing, or pushing an object. As prior data, the robot has access to an offline dataset of trajectories $\mathcal{D}_{\text{offline}} = \{\tau_1, \tau_2, \ldots, \tau_N\}$ for offline pre-training. In each trajectory, the robot is controlled by a human tele-operator or a scripted policy to achieve one of the goals the environment affords. A goal-conditioned policy is pre-trained on this dataset using offline RL algorithms.

After offline pre-training, the robot is placed in a particular environment that it has online access to interact in. Even though the initial configuration of this environment may have been included in the set of training environments, the goal distribution for this environment at test time requires sequencing multiple skills together, which is not present in the offline data. For instance, as shown in Fig. 1, the robot would need to first slide away the can that blocks the drawer, then reaches the handle of the drawer, and finally opens the drawer.

Naïvely running offline RL may not solve the long-horizon test tasks for two reasons. First, the robot is given a test goal distribution that is long horizon but offline dataset consists of individual skills. The method needs to somehow compose these individual skills autonomously in order to succeed at goals drawn from the test distribution. Second, offline RL may not solve the task due to distribution shift. Distribution shift appears in two forms: distribution shift between transitions in the prior data and transitions obtained by the actively rolling out the policy, and the distribution shift introduced when performing tasks sequentially. If the robot may actively interact in the new environment to improve its policy, how can the robot further practice and improve its performance?

3 PRELIMINARIES

We consider a goal-conditioned Markov Decision Process (MDP) denoted by a tuple $M = (\mathcal{S}, \mathcal{A}, \rho, P, \mathcal{G}, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, initial state probability ρ, transition probability P, a goal space $\mathcal{G}$, and discount factor γ. In each episode, a desired goal $s_g \in \mathcal{G}$ is sampled for the robot to reach. At each time step t, a goal-conditioned policy $\pi(a_t|s_t, s_g)$ selects an action $a_t \in \mathcal{A}$ conditioned on the current state s_t and goal s_g. After each step, the robot receives the goal-reaching reward $r_t(s_{t+1}, s_g)$. The robot aims to reach the goal by maximizing the average cumulative reward $\mathbb{E}[\Sigma_t \gamma^t r_t]$. Our approach learns a goal-conditioned policy π for solving the target task specified by a desired final goal s_g. The goal-conditioned policy is pre-trained on a previously collected offline dataset $\mathcal{D}_{\text{offline}}$ and then fine-tuned to reach s_g by accumulating data into an online replay buffer $\mathcal{D}_{\text{online}}$. $\mathcal{D}_{\text{offline}}$ contains diverse short-horizon interactions with objects in the environment. During online fine-tuning, we would like the policy to learn to improve and compose these short-horizon behavior for multi-stage tasks specified by s_g.

Defining informative goal-reaching rewards and extracting useful state representations from high-dimensional raw observations such as images can be challenging. Following the practice in prior work (Nair et al., 2018; Khazatsky et al., 2021), we pre-train a state encoder $h = \phi(s)$ to extract the latent state representation h. By encoding the states and goals to the latent space, we can obtain an informative goal-reaching reward function $r_t = R(h_{t+1}, h_g)$ by computing $h_{t+1} = \phi(s_{t+1})$ and $h_g = \phi(s_g)$. Specifically, $R(h_{t+1}, h_g)$ returns 0 when $||h_{t+1}, h_g|| < \epsilon$ and -1 otherwise, where ϵ is a selected threshold. In addition, we also use $\phi(s_{t+1})$ as the backbone feature extractor in all of our models that take s as an input. For simplicity, we directly use s to denote h in the rest of the paper. The details of the state encoder are explained in Sec. A.

4 PLANNING TO PRACTICE

We propose Planning to Practice (PTP), an approach that efficiently fine-tunes a goal-conditioned policy to solve novel tasks. To enable the robot to efficiently learn to solve the target task, we propose

to to use subgoals to facilitate the online fine-tuning of the goal-conditioned policy. Given the initial state s_0 and the goal state s_g, we search for a sequence of K subgoals $\hat{s}_1 : K = \hat{s}_1, ..., \hat{s}_K$ to guide the robot to reach s_g. Such subgoals will inform the goal-conditioned policy $\pi(a|s, s_g)$ what is the immediate next step on the path to s_g and provide the policy more dense reward signals compared to directly using the final goal. We choose the sequence of subgoals at the beginning of each episode and feed the first subgoal in the sequence to the goal-conditioned policy. The policy will switch to the next subgoal in the sequence when the current subgoal is reached or the time budget assigned for the current subgoal runs out.

The main challenge is to search for a sequence of subgoals that can lead to the desired final goals while ensuring each subgoal is a valid state that can be reached from the previous subgoal. Particularly when the states correspond to full images, most vectors will not actually represent valid states, and indeed naïvely optimizing over image pixels may simply result in out-of-distribution inputs that lead to erroneous results when input into the goal-conditioned policy.

As outlined in Fig. 1, we devise a method to effectively propose and select valid subgoal sequences to guide online fine-tuning by means of a generative model. At the heart of our approach is a conditional subgoal generator $g(\cdot|s_0)$ that recursively produces candidate subgoals in a hierarchical manner conditioned on the initial state s_0. To find the optimal sequence of subgoals $\hat{s}^*_{1:K}$, we first sample N candidate sequences $\hat{s}^1_{1:K}, ..., \hat{s}^N_{1:K}$ from the state space using the conditional subgoal generator. Then we rank the candidate sequences using a cost function $c(s_0, \hat{s}_{1:K}, s_g)$. The sequence that corresponds to the lowest cost will be selected as $\hat{s}^*_{1:K}$ for the goal-conditioned policy. Through this sampling-based planning procedure, we choose the subgoal for guiding the goal-conditioned policy π during online fine-tuning. The overall algorithm is summarized in Algorithm 1. Next we describe the design of each module in details.

Algorithm 1 Planning To Practice (PTP)

Require: set of final goals $\mathcal{G}$, time horizon T, offline data $\mathcal{D}_{\text{offline}}$, number of subgoals K.
Train $\pi(a|s, s_g)$ and $g(s, z)$ on $\mathcal{D}_{\text{offline}}$.
Initialize the online replay buffer $\mathcal{D}_{\text{online}} \leftarrow \varnothing$.
while not converged **do**
  Reset the environment and observe s_0.
  Sample s_g from $\mathcal{G}$.
  Plan for the subgoals $\hat{s}_{1:K}$.
  $k \leftarrow 1$
  for $t = 1, ..., T$ **do**
    Compute the action $a_t \leftarrow \pi(a_t|s_t, \hat{s}_K)$
    Observe the state s_{t+1} and the reward r_t
    $\mathcal{D}_{\text{online}} \leftarrow \mathcal{D}_{\text{online}} \cup (s_t, a_t, r_t, s_{t+1})$.
    if $t \pmod{\Delta t} == 0$ **or** $||s_{t+1} - \hat{s}_K|| < \epsilon$ **then**
      $k \leftarrow \min(k + 1, K)$
    end if
  end for
  Train π on batches sampled from $\mathcal{D}_{\text{offline}}$ and $\mathcal{D}_{\text{online}}$.
end while

Algorithm 2 $Plan(s_0, s_g, L, K, M, N)$

Require: the initial state s_0, the goal state s_g, number of subgoals K, number of levels L, multiplier M, number of samples N.
Sample N latent action sequences $\{z^i_{1:K}\}^N_{i=1}$.
Recursively generate subgoals $\{\hat{s}^i_{1:K}\}^N_{i=1}$ using $g(s, z)$.
Select $z^*_{1:K}$ and $\hat{s}^*_{1:K}$ of the lowest cost.
Update $z^*_{1:K}$ and $\hat{s}^*_{1:K}$ using MPPI.
if L = 1 **then**
  return $\hat{s}^*_{1:K}$
else
  Denote $\hat{s}^*_0 \leftarrow s_0$.
  Initialize the plan $\hat{\mathcal{S}}$ as an empty list
  for $i = 1, ..., K$ **do**
    Append $Plan(\hat{s}^*_{i-1}, \hat{s}^*_i, L - 1, M, M, N)$ to $\mathcal{S}$
  end for
  return $\hat{\mathcal{S}}$
end if

4.1 Conditional Subgoal Generation

The effectiveness of our planner relies on the generation of diverse and feasible sequences of subgoals as candidates. Specifically, we would like to generate the candidates by sampling from the distribution of suitable subgoal sequences $p(\hat{s}_1, ..., \hat{s}_K|s_0)$ conditioned on the initial state s_0. Most existing methods independently sample the subgoal at each step from a learned prior distribution (Pertsch et al., 2020) or a replay buffer (Eysenbach et al., 2019), which is unlikely to propose useful plans for tasks with large, combinatorial state spaces (i.e., with multiple objects).

We propose to break down $p(\hat{s}_1, ..., \hat{s}_K|s_0)$ into $p(\hat{s}_1|s_0)\Pi_{i=1}^{k} p(\hat{s}_i|\hat{s}_{i-1})$ through modeling the conditional distribution $p(s'|s)$ of the reachable next subgoal s'. By utilizing temporal compositionality, the conditional subgoal generation paradigm improves generalization and enables generation of sequences of arbitrary lengths.

We use a conditional variational encoder (CVAE) (Sohn et al., 2015) to capture the distribution of reachable goals $p(s'|s)$. In the CVAE, we define the decoder as $g(s, z)$ and the encoder as $q(z|s, s')$, where z is the learned late representation of the transitions and it is sampled from a prior probability $p(z)$. To propose a sequence of subgoals, we use $g(s, z)$ as the conditional subgoal generator. Conditioned on the initial state s_0, the first subgoal $\hat{s}_1$ can be generated as $\hat{s}_1 = g(s_0, z_1)$ given the sampled z_1. Then the i_{th} subgoal can be recursively generated by sampling $z_i \sim p(z)$ and computing $\hat{s}_i = g(\hat{s}_{i-1}, z_i)$ given the previous subgoal $\hat{s}_{i-1}$. In this way, we could sample a sequence of i.i.d. latent representations $z_1, ..., z_K$ and recursively generate $\hat{s}_1, ..., \hat{s}_K$ conditioned on the initial state s_0 using the conditional subgoal generator.

The CVAE is trained to minimize the evidence lower bound (ELBO) (Kingma & Welling, 2014) of $p(s'|s)$ given the offline dataset $\mathcal{D}$. During training, we sample transitions (s_t, s_τ) from the offline dataset to form the minibatches, where $\tau = t + \Delta t$ is a future step that is Δt steps ahead. Instead of using a fixed Δt, we sample Δt from a range for each transition to provide richer data. To encourage the trained model to be robust to compounding errors, we sample sequences composed of multiple states and use the subgoal reconstructed at the previous step as the context in the next step. Therefore, the objective for training the conditional subgoal generator is:

$$\mathbb{E}_{q(z|s_t,s_\tau)}||s_\tau - g(s_t, z)||^2 + D_{KL}[q(z|s_t, s_\tau)||p(z)] \tag{1}$$

where $D_{KL}[\cdot||\cdot]$ indicates the KL-Divergence.

4.2 Efficient Planning in the Latent Space

We build a planner that efficiently searches for sequences of subgoals in the latent space as shown in Algorithm 2. To tackle the large search space of candidate subgoal sequences, we design a hierarchical planning algorithm that searches for subgoals in a coarse-to-fine manner and re-use the previously selected subgoals as candidates in new episodes.

The hierarchical planning is conducted at L levels with different temporal resolutions Δt_1, ..., Δt_L. The temporal resolution of each level is an integral multiple of that of the previous level, i.e., $\Delta t_i = M\Delta t_{i-1}$, where M is a scaling factor and is set to 2 in our experiments. We first plan for the subgoals $\hat{s}_1^1, \hat{s}_2^1, ...$ on the first level. Then the subgoals $\hat{s}_{1:K}^l$ of finer temporal resolution are planned on each level l to connect the subgoals planned on the previous level $l-1$. Specifically, given the adjacent subgoals $\hat{s}_i^{l-1}$ and $\hat{s}_{i+1}^{l-1}$ produced on the previous level, we plan for a segment of M subgoals $\hat{s}_{i*M+1}^l, ..., \hat{s}_{(i+1)*M}^l$ on the level l, by treating $\hat{s}_i^{l-1}$ and $\hat{s}_{i+1}^{l-1}$ as the initial state and final goal state in Eqn. 3. The planned segments are returned to the previous level and concatenated as a more fine-grained plan. For this purpose, we train L conditional subgoal generators to propose subgoals that are Δt_1, ..., Δt_L steps away, respectively. In contrast to the prior work (Pertsch et al., 2020), the conditional subgoal generators enable us to plan for unseen goals that are beyond the temporal horizon of the demonstrations in the offline dataset by exploiting the compositional structure of the demonstrations. By recursively generating the subgoals across time at each level, we only need to enforce that the temporal resolution of the top level Δt_L is smaller than since the the conditional subgoal generator $f^1(s, z)$ needs to be trained on trajectories at least $\Delta t_L + 1$ steps in length.

We maintain a latent plan buffer for each level to further facilitate the planning with the conditional subgoal generator. After each episode, the selected latent representations on each level are appended to the corresponding latent plan buffer. In each target task, the subgoals are supposed to have the same semantic meaning. In spite of the variations of the initial and goals state in each episode, the optimal plans in the latent space can often be similar to each other. Therefore, we sample half of the latent representations from the prior distribution $p(z)$ and the other half from the latent plan buffer among the initial samples to enhance the chance of finding a close initial guess.

We build our planner upon the model predictive path integral (MPPI) (Gandhi et al., 2021), which iteratively optimizes the plan through importance sampling. In each interaction, we perturb the chosen plan in the latent space with a small Gaussian noise as new candidates.

4.3 COST FUNCTION FOR FEASIBLE SUBGOALS

To provide informative guidance to the policy $\pi(a|s, s_g)$, we would like that the final goal s_g can be reached at the end of the episode while encouraging the transition between each pair of subgoals to be feasible within a limited time budget. As explained in Sec. 3, the goal state is considered to be reached when the Euclidean distance between the last subgoal in the plan and the desired goal is less than a threshold δ in the learned latent space. The feasibility of each transition between adjacent subgoals can be measured using the goal-conditioned value function $V(s, s')$ trained by the reinforcement learning algorithm. Therefore, finding the subgoals $\hat{s}^*_{1:K}$ can be formulated as a constrained optimization problem:

$$\begin{aligned} &\text{minimize} && ||s_g - \hat{s}_K|| \\ &\text{subject to} && V(\hat{s}_i, \hat{s}_{i+1}) \geq \delta, \text{for } i = 0, ..., K-1 \end{aligned} \tag{2}$$

where we use $\hat{s}_0 = s_0$ to denote the initial state for convenience. By re-writing Eq. 2 as a Lagrangian, we obtain the cost function with a weight η:

$$c(s_0, \hat{s}_{1:K}, s_g) = ||s_g - \hat{s}_K|| + \eta \sum_{i=0}^{K-1} V(\hat{s}_i, \hat{s}_{i+1}) \tag{3}$$

The details of our method are explained in Appendix A.

5 EXPERIMENTS

In our experiments, we aim to answer the following questions: 1) Can PTP propose and select feasible subgoals as plans for real-world robotic manipulation tasks? 2) Can the subgoals planned by PTP facilitate online fine-tuning of the goal-conditioned policies to solve target tasks unseen in the offline dataset? 3) How does each design option affect the performance of PTP?

5.1 EXPERIMENTAL SETUP

Environment. As shown in Fig. 2, our experiments are conducted in a table-top manipulation environment with a Sawyer robot. At the beginning of each episode, a fixed drawer and two movable objects are randomly placed on the table. The robot can change the state of the environment by opening/closing the drawer, sliding the objects, and picking and placing objects into different destinations, etc. At each time step, the robot receives a 48 x 48 RGB image via a Logitech C920 camera as the observation and takes a 5-dimensional continuous action to change the gripper status through position control. The action dictates the change of the coordinates along the three axes, the change of the rotation, and the status of the fingers. We use PyBullet (Coumans & Bai, 2016–2021) for our simulated experiments.

Prior data. The prior data consists of varied demonstrations for different primitive tasks. In each demonstrated trajectory, we randomly initialize the environment and perform primitive interactions such as opening the drawer and poking the object. These trajectories are collected using teleoperation in the real world, and a scripted policy that uses privileged information of the environment (e.g., the object pose and the status of the drawer) in simulation. The trajectories vary in length from 30 to 100 time steps, with 1,000 trajectories in the real world and 4,000 in simulation.

Target tasks. In each target task, a desired goal state is specified by a 48 x 48 RGB image (same dimension with the observation). The robot is tasked to reach the goal state by interacting with the objects on the table. Task success for our evaluation is determined based on the object positions at the end of each episode (this metric is not used for learning). As shown in Fig. 2, we design three target tasks that require multi-stage interactions with the environment to complete. These target tasks are designed with temporal dependencies between stages (e.g.the robot needs to first move away a can that blocks the drawer before opening the drawer). The transitions from the initial state to the goal state are unseen in the offline data. The episode length is 400 steps in simulation and 250 steps in the real world, which are much longer than the time horizon of the demonstrations.

Baselines and ablations. We compare PTP with 3 baselines and 3 ablations. **Model-Free** uses a policy directly conditioned on the final goal and conducts online fine-tuning without using any subgoals. **LEAP** (Nasiriany et al., 2019) learns a variational auto-encoder (VAE) (Kingma & Welling, 2014) to capture the prior distribution of states and plans for subgoals without conditioning on any context. **GCP** (Pertsch et al., 2020) learns a goal predictor that hierarchically generates intermediate subgoals between the initial state and the goal state. To analyze the design options in PTP, we also compare with variations of our method by removing the latent plan buffer (**PTP (w/o B)**), the hierarchical planning algorithm (**PTP (w/o H)**), and both of these two designs (**PTP (w/o H and B)**). All methods use the same neural network architecture in the goal-conditioned policy and are pre-trained on the same offline dataset.

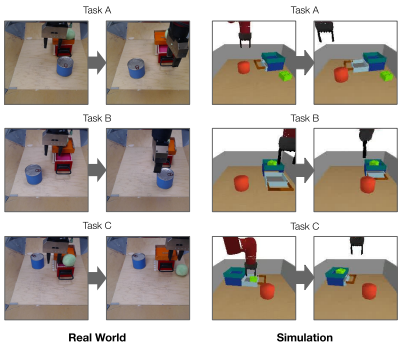

Figure 2: **Target tasks.** Three multi-stage tasks are designed for our experiments in the simulation and the real world respectively. In each target task, the robot needs to strategically interact with the environment (e.g.first take out an object in the drawer then close the drawer). The initial state and desired goal state are shown for each task.

5.2 Quantitative Comparisons

We evaluate PTP and baselines on three unseen target tasks. We use simulated versions of these tasks for comparisons and ablations, and real-world tasks, where all pretraining and finetuning uses only real-world data, to evaluate the practical effectiveness of the method.

Simulation. We first pre-train the goal-conditioned policy on the offline dataset for 100 epochs and the run online fine-tuning for the target task for 150 epochs. Each epoch takes 2,000 simulation steps (only during fine-tuning) and 2,000 training iterations. We run online fine-tuning using each method with 3 different random seeds. After each epoch, we test the policy in the target task for 5 episodes. We report the average success rate across 3 runs in Fig. 4 where the negative x-axis indicates the offline pre-training epochs and positive x-axis indicates the online fine-tuning epochs. As shown in Fig. 4, our full model consistently outperforms baselines with a large performance gap. The generated subgoals not only enables the pre-trained policy to achieve higher success rate by breaking down the hard problems into easier pieces, but also introduces larger performance improvements during online fine-tuning. After fine-tuning for 150 epochs, the policy achieves the success rates of 84.9%, 59.9%, 49.3% in the three target tasks respectively. Compared to the policy pre-trained on the offline dataset, the performance is significantly improved (+31.6%, +37.8%, and +13.8%). When directly using the final goal or subgoals generated by baseline methods, the policy's performance plateaus at around 0.0% to 30.0% and doesn't improve much during online fine-tuning. We found that the hierarchical planner and latent plan buffer are crucial for PTP's performance. Without these two design options, the planner often suffers from the large search space of possible subgoal sequences and the resultant success rates decrease. The latent plan buffer significantly improves the performance of non-hierarchical PTP and has a minor effect on hierarchical PTP.

Real-world evaluation. We pre-train the policy for 50 epochs and fine-tune it for 10 epochs. In each epoch, we run 1,000 training iterations and collect 1,000 steps in the real world. We train on three target tasks which are shown in Figure 2, and report the success rate of the goal-conditioned policy before and after online fine-tuning in Table 1. Planning enables the robot to succeed partially with just the offline initialized policy, achieving success rates of $25\%, 25\%$ and 12.5% on the three tasks. (When the offline policy is conditioned on only the final goal image without planning, the success rate is 0%.) Then in each task, we fine-tune to a significantly higher success rate. Qualitatively, at the begin-

Table 1: The real-world success rates before and after online fine-tuning. The tasks are described in Sec. 5.1.

Task	PTP (Ours) Offline → Online	GCP Offline → Online
Task A	$25.0\% \rightarrow$ **62.5%**	$0.0\% \rightarrow 0.0\%$
Task B	$12.5\% \rightarrow$ **62.5%**	$0.0\% \rightarrow 0.0\%$
Task C	$12.5\% \rightarrow$ **100%**	$12.5\% \rightarrow 37.5\%$

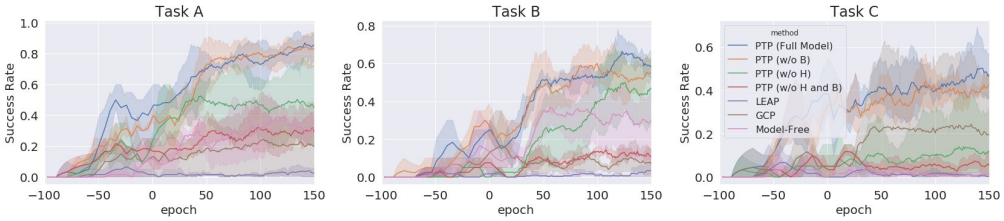

Figure 4: **Quantitative comparison in simulation.** The average success rate across 3 runs is shown with the shaded region indicating the standard deviation. The negative x-axis indicates the epochs of offline pre-training and positive x-axis indicates epochs of online fine-tuning. Using offline learning and planning, our method PTP is able to solve these tasks partially (at 0 epochs). Then with online finetuning the performance improves further. In contrast, prior methods have lower offline performance and do not fine-tune successfully in most cases, as they do not collect coherent online data.

ning of fine-tuning, the robot often fails, deviating from the planned subgoals or colliding with the environment. With the planned subgoals, the original long-horizon task is broken down to short snippets that are easier to complete. Even if a subgoal is not reached successfully at first, the data is useful to collect additional experience and fine-tune the policy. After fine-tuning for 4-5 epochs, we already observe that the robot's performance reaching subgoals during training time significantly improves, collecting even more coherent and useful data. After 10 epochs, we achieve success rates of 62.5%, 62.5% and 100%. In comparison, GCP cannot provide useful guidance to the policy when the generated goals are noisy.

5.3 Generated Subgoals

In Fig. 3, we present qualitative results of the generated subgoals for each task in the real world. Each row shows a sequence of generated subgoals produced by the planner in each method. In all the three target tasks, PTP successfully plans for a sequence of subgoals that can lead to the desired final goal. The transition between adjacent subgoals are feasible within a short period of time. By comparison, both of the baseline methods fail to generate reasonable plans. Without conditioning on the current state, LEAP (Nasiriany et al., 2019) can hardly produce any realistic images of the environment. Most of the generated subgoals are highly noisy images with duplicated robot arms and objects. The quality of the subgoals produced by GCP (Pertsch et al., 2020) is higher than that of LEAP but still much worse than ours. GCP cannot generalize well for the initial state and the goal state that are out of the distribution of the offline dataset, which contains only short snippets of demonstrations.

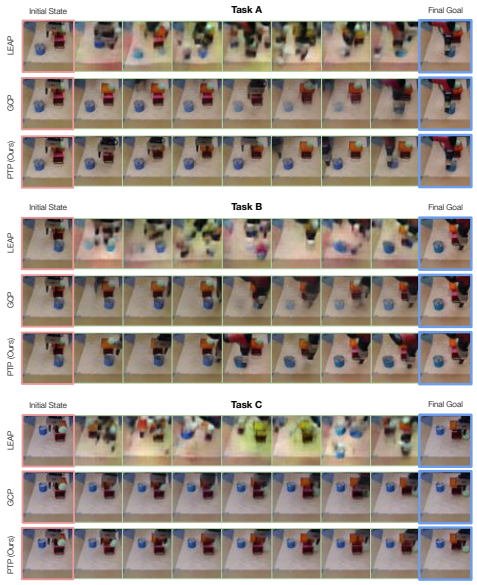

Figure 3: **Planned subgoal sequences.** Each row shows the sequence of subgoals produced by each method. The initial state and the final goal are shown at the two ends.

6 Conclusion and Discussion

We presented PTP, a method for real-world learning of temporally extended skills by utilizing planning and fine-tuning to stitch together skills from prior data. First, planning is used to convert a long-horizon task into achievable subgoals for a lower level goal-conditioned policy trained from prior data. Then, the goal-conditioned policy is further fine-tuned with active online interaction, mitigating the distribution shift between the offline data and actual states seen during rollouts. This procedure allows robots to extend their capabilities autonomously, composing previously seen data into more complicated and useful skills.

REFERENCES

Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Joshua Tobin, P. Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, 2017.

Yevgen Chebotar, Karol Hausman, Yao Lu, Ted Xiao, Dmitry Kalashnikov, Jake Varley, Alex Irpan, Benjamin Eysenbach, Ryan Julian, Chelsea Finn, and Sergey Levine. Actionable models: Unsupervised offline reinforcement learning of robotic skills. 4 2021.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.

Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In *NeurIPS*, 2019.

Manan S. Gandhi, Bogdan Vlahov, Jason Gibson, Grady Williams, and Evangelos A. Theodorou. Robust model predictive path integral control: Analysis and performance guarantees. *IEEE Robotics and Automation Letters*, 6:1423–1430, 2021.

Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. 2021.

Alexander Khazatsky, Ashvin Nair, Dan Jing, and Sergey Levine. What can i do here? learning new skills by imagining visual affordances. *ICRA*, pp. 14291–14297, 2021.

Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pp. 1097–1105, 2012.

Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. *International Journal of Robotics Research*, 2017.

Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual Reinforcement Learning with Imagined Goals. In *NeurIPS*, 2018.

Soroush Nasiriany, Vitchyr H. Pong, Steven Lin, and Sergey Levine. Planning with goal-conditioned policies. In *NeurIPS*, 2019.

Karl Pertsch, Oleh Rybkin, Frederik Ebert, Chelsea Finn, Dinesh Jayaraman, and Sergey Levine. Long-horizon visual planning with goal-conditioned hierarchical predictors. *ArXiv*, abs/2006.13205, 2020.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

Kihyuk Sohn, Xinchen Yan, and Honglak Lee. Learning Structured Output Representation using Deep Conditional Generative Models. In *Advances in Neural Information Processing Systems*, 2015.

Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *NeurIPS*, 2017.

Tianhe Yu, Aviral Kumar, Yevgen Chebotar, Karol Hausman, Sergey Levine, and Chelsea Finn. Conservative data sharing for multi-task offline reinforcement learning. *NeurIPS*, 34, 12 2021.

A IMPLEMENTATION DETAILS

Following Khazatsky et al. (2021), we use a vector quantized variational autoencoder (VQ-VAE) (van den Oord et al., 2017) as the state encoder, which encodes a $48 \times 48 \times 3$ image to a 720-dimensional encoding. We use the same network architectures for the policy and the value functions from Khazatsky et al. (2021). The conditional subgoal generator is implemented with a U-Net architecture (Ronneberger et al., 2015) and decodes the subgoal from a 8-dimensional latent representations conditioned on the encoding of the current state. In our planner, we use $L = 3$, $K = 8$, $M = 2$, $N = 1024$, and we run MPPI for 5 iteration on each level. g is trained to predict subgoals that are 15, 30, and 60 steps away. Implicit Q-Learning (IQL) (Kostrikov et al., 2021) is used as the underlying RL algorithm for offline pre-training and online fine-tuning with default hyperparameters. We use Adam optimizer with a learning rate of $3 \cdot 10^{-4}$ and a batch size of 1024. During training, we relabel the goal with future hindsight experience replay (Andrychowicz et al., 2017) with 70% probability. We use $\epsilon = 3$ for the reward function defined in Sec. 3, $\eta = 0.01$ in Eqn. 3.