2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Demonstration-Bootstrapped Autonomous Practicing via Multi-Task Reinforcement Learning

Abhishek Gupta[1], Corey Lynch[2], Brandon Kinman[2], Garrett Peake[2], Sergey Levine[1] and Karol Hausman[2]

Abstract— **Reinforcement learning systems have the potential to enable continuous improvement in unstructured environments, leveraging data collected autonomously. However, in practice these systems require significant amounts of instrumentation or human intervention to learn in the real world. In this work, we propose a system for reinforcement learning that leverages multi-task reinforcement learning bootstrapped with prior data to enable continuous autonomous practicing, minimizing the number of resets needed while being able to learn temporally extended behaviors. We show how appropriately provided prior data can help bootstrap both low-level multi-task policies and strategies for sequencing these tasks one after another to enable learning with minimal resets. This mechanism enables our robotic system to practice with minimal human intervention at training time, while being able to solve long horizon tasks at test time. We show the efficacy of the proposed system on a challenging kitchen manipulation task both in simulation and the real world, demonstrating the ability to practice autonomously in order to solve temporally extended problems.**

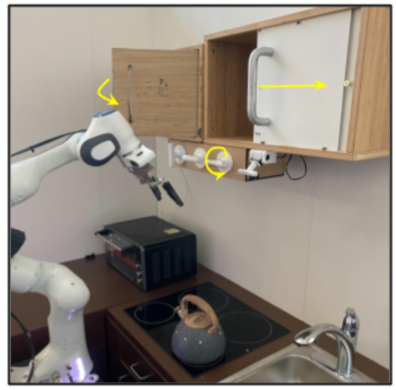

Fig. 1: Robot setup for real world training in a kitchen. The robot needs to manipulate multiple different elements to accomplish complex goals.

I. Introduction

Consider a robot deployed in a previously unseen kitchen, such as the one shown in Fig. 1. Since this is an unfamiliar environment, the robot might not know exactly how to operate all the kitchen appliances, open cabinets, operate the microwave or turn on the burners, especially if these need to be done in a specific sequence resulting in a long-horizon task. Like many intelligent agents, the robot needs some practice, making reinforcement learning (RL) a promising paradigm. However, as we have learn from prior work, practicing long-horizon tasks with RL is difficult for reasons such as exploration [1], environment resets [2], and credit assignment problems [3]. The process of building robotic learning systems (especially with RL) is never as simple as placing a robot in the environment and having it continually improve. Instead, human intervention is often needed in the training process, especially to provide resets to the learning agent. Ideally, we would want to provide the robot with a small number of examples to help with these challenges in the training process and do so at the beginning of the practicing process so that it can continually practice a wide range of tasks in the kitchen on its own, with minimal human intervention.

In this work, we consider the following perspective on this problem. Rather than only providing examples that aim to aid the final solution (e.g. by providing demonstrations of the final long-horizon tasks), we additionally focus on providing supervision that helps the robot to continue practicing long-horizon tasks autonomously. In particular, we provide a set of continuous, uninterrupted multi-task demonstrations [4], [1] that illustrate example behaviors and goal states that the robot should be able to acquire and robustly achieve. The continuous aspect of the demonstrations provides two benefits: i) it breaks up the long-horizon tasks into smaller subgoals that can be used as intermediate points, and ii) it provides data *in-between* the tasks, showing the robot examples of behaviors where some of the tasks are not fully finished but are reset using other tasks - a case that often happens during the autonomous execution of the policy. Equipped with the continuous demonstrations, we can learn how to sequence different tasks one after the other and, use that ability to have some tasks provide resets for others, enabling the robot to practice many tasks autonomously by resetting itself without actually needing constant human intervention in the learning process. For instance, if the robot is practicing opening and closing the cabinet in Fig. 1, it may start in a state where the cabinet is closed and command the task to open it. Even if this task does not succeed and the cabinet is partially open, the agent can continue practicing any of the remaining tasks including closing or opening the cabinet to minimize the requirement for resets, naturally resetting the environment by just solving different tasks. At test time, we can then utilize the same ability of sequencing different tasks to sequence multiple subgoals for completing long-horizon multi-stage tasks.

In this paper, we present a novel robotic learning system, which we call demonstration bootstrapped autonomous practicing (DBAP), that is able to incorporate a small amount of human data provided upfront to bootstrap long periods of

[1]These authors are with UC Berkeley. Correspondence to: {abhigupta, svlevine} @eecs.berkeley.edu
[2]These authors are with Robotics at Google.

autonomous learning with just a few human interventions for episodic resets, using some tasks to reset others while learning how to perform long horizon behaviors. We show that just a few hours of easily-provided "play-style" [4], [1] multi-task demonstrations can make the entire learning process more efficient, bootstrapping both policy learning and autonomous practicing of behaviors, eventually showing the ability to perform long-horizon multi-step tasks. Our experiments indicate that this paradigm allows us to efficiently solve complex multi-task robotics problems both in simulation and in a real world kitchen environment, with an order of magnitude less human intervention needed during learning than typical RL algorithms.

II. Related Work

Most prior applications of RL in robotic manipulation rely on an episodic reset mechanism, which brings the robot back to a static distribution over initial states. Recent work has attempted to relax the episodic reset assumption [5], [6], [7], [2] by either learning an explicit reset [7], [2] or a perturbation controller [6] to reset the state. As opposed to these works, we operate in the continual multi-task setting. Recent work [8], [9], [2] has approached the problem through the lens of multi-task learning, but require human effort to design how tasks should be sequenced, relying on human-defined state machines. In contrast, our work shows how we can bootstrap the autonomous learning process with human-provided prior data, obviating the need for definition of task sequencing state-machines.

There are multiple approaches that use demonstrations to provide a good initialization for policies via learning from demonstrations [10], [11], [12], [13], [14], [15]. In a similar vein, algorithms for offline RL [16] use large datasets to bootstrap learning, without making explicit assumptions about the provided data. Most algorithms for offline RL aim to enable training from static offline datasets, by preventing the off-policy bootstrap from diverging due to internal distribution shift [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26]. While most of these are in purely offline settings, recently there have been approaches [23], [27] that bootstrap initial policies offline but aim to also continue finetuning. In contrast to these papers, which utilize offline data simply as an initialization for a task policy that can be finetuned online but assume complete access to *episodic resets*, our aim is to instead utilize the provided data to learn how to perform *autonomous practicing*: for our method the prior data doesn't just provide a good initialization for a single task, but a set of useful behaviors that the agent can use to reset itself after failed attempts, and chain together to perform multi-stage behaviors.

Long horizon problems have been approached with hierarchical RL algorithms [28], [29], [30], [31], [32], [1], [33], [34], [35], [36], [37]. These algorithms learn high-level policies that sequence low-level policies to solve temporally-extended tasks. In addition to sequencing tasks for long-horizon goals, our system is also used to perform autonomous practicing with minimal human interventions. Most closely related to our work, relay policy learning (RPL) [1], uses demonstrations to bootstrap a hierarchical learning setup. However, RPL assumes episodic resets at training time, which DBAP alleviates.

III. Preliminaries and Problem Statement

We use the standard MDP formalism $\mathcal{M} = (S, A, P, r, T)$ (state, action, dynamics, reward, episodic horizon), but extend RL to goal-reaching tasks, where the task is represented as a state goal s_g that the goal-conditioned policy $\pi_\theta(a|s, s_g)$ has to reach to obtain the reward. We assume to be given a set of N goal states that are of particular interest: $S_{g_i} = \{s_g^i\}_{i=1}^N$ that the agent is expected to to reach from any other state of interest, resulting in the following objective: $\max_\theta \mathbb{E}_{s_0=s_g^i, a_t\sim\pi_\theta(a_t|s_t,s_g^j)} \sum_{t=1}^T r(s_t, a_t, s_g^j) \quad \forall s_g^i \in S_{g_i}, s_g^j \in S_{g_i}$

We often refer to reaching these goals of interests as *tasks* since reaching such states usually corresponds to accomplishing a meaningful task such as opening a cabinet. Note that reaching certain s_g^j from s_g^i would involve just short horizon behaviors spanning a single episode, while reaching other s_g^j from s_g^i may require long horizon reasoning over multiple different sub-steps, for instance it may require reaching $s_g^k, s_g^l, ...$ on the path from s_g^i to s_g^j.

In order to allow for more autonomous learning with minimal human supervision, we assume that a reset can be provided every n episodes (or every nT time steps). Our objective in this case is to train the agent in a training MDP $\mathcal{M}' = (S, A, P, r, nT)$, where resets are only received every nT steps such that we are able to learn a policy π that is effective on the original episodic problem described above. This requires our learning algorithm to operate in a non-stationary setting and learn how to practice tasks mostly autonomously with only occasional resets [7], [8], [6].

AWAC: As we will discuss next, solving this problem effectively for long horizon tasks requires us to incorporate demonstrations to bootstrap both the policy and the autonomous practicing mechanism. In order to effectively incorporate prior demonstrations to bootstrap policy learning, we use the recently developed AWAC algorithm [23]. The main idea behind AWAC is to constrain the actor policy to be within the training data distribution, which addresses the problem of overly optimistic Q-function estimates on unseen state-action pairs [18], [20]. In particular, the AWAC algorithm applies the following weighted maximum likelihood update to the actor: $\theta_{k+1} = \arg\max_\theta \mathbb{E}_{s,a} \left[\log \pi_\theta(a|s) \exp\left(\frac{1}{\lambda} A^{\pi_\theta}(s,a)\right)\right]$, where is A is the advantage function, and λ is a hyper-parameter. Since the actor update in AWAC samples and re-weights the actions from the previous policy, it implicitly constrains the policy. We directly build on this framework for policy bootstrapping but in the following section show how we can additionally bootstrap autonomous practicing from offline data as well.

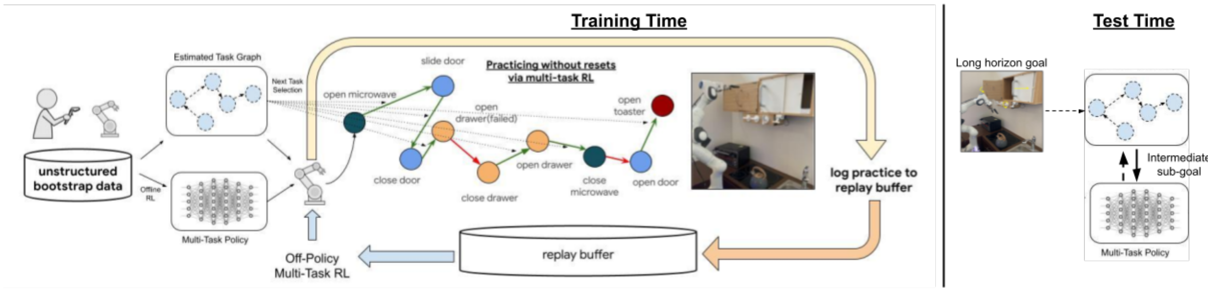

Fig. 2: Given human-provided unstructured demonstrations, the system bootstraps a multi-task RL policy via offline RL and builds a task graph that models transitions between different tasks. The system then practices the tasks autonomously with small number of resets, using the task graph to command the appropriate next task. The resulting multi-task policy and task graph are then used to solve long-horizon problems at test-time.

IV. Demonstration Bootstrapped Autonomous Practicing for Multi-Task Reinforcement Learning

Our method, Demonstration-Bootstrapped Autonomous Practicing (DBAP) learns how to perform complex behaviors in an environment by leveraging human-provided demonstration data in two ways: to bootstrap a multi-task policy that learns to perform a variety of behaviors; and to enable a graph-based sequencing mechanism, which allows for autonomous improvement by choosing what sequence of these behaviors to command to minimize number of resets at training time. In doing so, this sequencing mechanism helps to practice each behavior at training time while also enabling the robot to sequence practiced behaviors to accomplish long-horizon tasks at test time. An overview of our overall framework is presented in Fig. 2. Given a human-provided dataset, we leverage an offline RL algorithm to bootstrap a multi-task policy, while using a graph search algorithm to i) decide how to sequence tasks for practicing to improve this multi-task policy with minimal resets at training time and ii) deciding how to sequence behaviors to solve long-horizon tasks at test-time.

More precisely, we learn a single goal-conditioned policy for multiple tasks, denoted $\pi_\theta(a|s, s_g)$, where s_g is a goal state. To learn this policy, we instantiate a goal-conditioned version of AWAC [23], where the policy is provided with the state goal s_g alongside the current state s, yielding the following policy update: $\theta_{k+1} = \arg\max_\theta \mathbb{E}_{s,a,s_g}\left[\log \pi_\theta(a|s, s_g)\exp\left(\frac{1}{\lambda}A^{\pi_\theta}(s, a, s_g)\right)\right]$, where A^{π_θ} is an advantage function for the current policy π_θ. This goal-conditioned policy is directly bootstrapped from human provided data as described in prior work [23] Prior data is not just used for policy bootstrapping but to autonomously improve this bootstrapped multi-task policy through autonomous practicing with minimal numbers of human resets, while also being able to leverage the policy and practicing mechanism appropriately to solve long horizon problems.

In order to enable agents to practice on their own with minimal number of human interventions, we require a task-sequencing policy that determines how the goals are sequenced at training time. In particular, a task-sequencing policy $\pi^{\text{hi}}(s_g|s, s_g^{\text{desired}})$ decides which goal of interest s_g to command next from the current state during practicing. This resembles high-level policies used in hierarchical RL that are commonly used to sequence individual subgoals for accomplishing long-horizon goals. However, in this scenario, $\pi^{\text{hi}}(s_g|s, s_g^{\text{desired}})$ is not just used to accomplish long-horizon goals at test-time but also to enable autonomous practicing at training time. In the following sections we will often refer to the multi-task policy $\pi_\theta(a|s, s_g)$ as the low-level policy, and the task sequencer $\pi^{\text{hi}}(s_g|s, s_g^{\text{desired}})$ as the high-level policy.

Assumptions: Firstly, we assume that the environment does not end up in irrecoverable states from which no action can bring the environment back to goals of interest. This assumption is common across a variety of reset-free RL algorithms [8], [2], [6], [7], [9] and spans a wide range of practical tasks in robotic manipulation and locomotion. We also assume that the states can be discretely grouped into different tasks which in our case is done by a thresholding operation (details in the Appendix).

Task Sequencing via Graph Search. A successful high-level task sequencer policy π^{hi} is one that at training time is able to propose goals for autonomous practicing, while at test time is able to direct the agent towards reaching long-horizon goals. To learn such a task sequencer policy π^{hi}, we propose a simple model-based graph search algorithm. Note that π^{hi} is not a neural network, but the result of a search procedure. As we show empirically in our experimental evaluation, this is more flexible and robust than parameterizing π^{hi} directly with a neural network.

The key idea in our approach is to leverage prior data to learn which low-level task transitions are *possible* and can be sequenced, and then use this knowledge to optimize for *both* autonomous practicing (at training time) and long-horizon goal sequencing (at test time). In particular, we utilize the provided data to build a directed task graph $\mathcal{G}$, with vertices as different goal states of interest s_g^i, s_g^j, and an adjacency matrix A with $A(i, j) = 1$ if there exists a transition between particular states of interest s_g^i and s_g^j in the demonstration data, and $A(i, j) = 0$ otherwise. This graph can be thought of as a discrete high-level model, which represents how different goal states of interest are connected to each other. Given this graph G acquired from the prior data, we can

Algorithm 1 PracticeGoalSelect: High Level Task Selection via Graph Search

Input: task-graph $\mathcal{G}$, goal-states $\{s_g^0, .., s_g^N\}$, density over goals $\rho(s_g^i)$, current state s

Output: Next goal s_g^{next} to command

```
// Initialize maximum entropy value
H_max = -∞
for s_g^i ∈ {s_g^1, ..., s_g^N} do
    τ ← Dijkstra(s, s_g^i, G)
    ρ' = ρ + τ // Compute updated density
    if H(ρ') ≥ H_max then
        H_max = H(ρ')
        s_g^next ← τ[1] // First step of path to s_g^i
    end if
end for
return s_g^next
```

Algorithm 2 Overview of DBAP

Input: Human provided multi-task demonstrations $\mathcal{D}$

Output: Multi-task policy $\pi_\theta(a|s, s_g)$ and task-graph $\mathcal{G}$

```
Estimate task graph G via counting
Initialize π_θ(a|s, s_g) via AWAC [23]
Initialize current density ρ = 0
for t = 0, ..., N steps do
    // Select next goal via graph search
    s_g^next ← PracticeGoalSelect(s, ρ)
    Rollout π_θ(a|s, s_g^next)
    Add collected data to replay buffer β
    Update π_θ via AWAC.
end for
return π_θ(a|s, s_g), G
```

then utilize it for both autonomous practicing of low-level tasks and for sequencing multiple low-level tasks to achieve multi-step goals. We describe each of these phases next.

Autonomous practicing (training time). As described in prior work [6], maintaining a uniform distribution over tasks helps when learning with minimal resets by ensuring all tasks are equally represented in the learning process. The task graph $\mathcal{G}$ is used to direct autonomous practicing by explicitly optimizing to command goals that bring the overall coverage over states close to the uniform distribution, while also minimizing the number of human interventions.

In particular, the algorithm iterates through all possible goal states of interest s_g^i, determines the shortest path from the current state to the goal state via Dijkstra's algorithm [38], and computes the resulting densities over goal states if that path was chosen. The path that results in bringing the density closest to uniform (i.e. maximum entropy) is then picked, and the first step in the path (i.e. a short-horizon goal) is commanded as the next goal state. This ensures that the algorithm maintains uniform coverage over different goal states of interest in the environment, so that all tasks can be practiced equally. This process repeats at the next step, thereby performing receding horizon control [39] to maintain the density over potential goal states of interest to be as close to uniform as possible, while training the policy to keep practicing how to accomplish various short horizon goals.

Formally, the objective being optimized to select which goal to sequence next is given by: $\max_{s_g^i \in S_g^i} \mathcal{H}(\mathcal{U}, \rho + \text{Djikstra}(s, s_g^i))$, where ρ is the current marginal distribution (represented as a categorical distribution) over goal states of interest, Djikstra(s, s_g^i) computes the shortest distances between current state s and s_g^i and the goal is to bring the updated density as closer to uniform $\mathcal{U}$[1]. A detailed description can be found in Algorithm 1.

Task sequencing for long-horizon tasks (test time). Next, we ask how short-term behaviors learned by the multi-task policy $\pi_\theta(a|s, s_g)$ can be sequenced to accomplish long-horizon goals. We note that the same graph search mechanism used for autonomous practicing at training time can be reused to sequence appropriate sub-goals at test time to accomplish long horizon tasks.

More formally, given a target goal of interest s_g^j when the agent is at a particular state s, we can use the estimated graph $\mathcal{G}$ via Dijkstra's algorithm to compute the shortest path between the current state and the goal of interest s_g^j ($\tau = [s, s_g^1, s_g^2, \ldots, s_g^j]$). The next goal state of interest in the path s_g^1 is then chosen as the next goal commanded to the multi task policy $\pi(a|s, s_g^1)$ and executed for a single episode of T steps till the agent reaches s_1. This procedure is repeated until the agent accomplishes s_g^j. Further details on these procedures can be found in Algorithm 1 and Algorithm 2.

V. System Description

To evaluate (DBAP) in the real world, we built a physical kitchen environment based on kitchen manipulation benchmark described in prior work [1].

Tasks. In this environment, we task a 7 DoF Franka Emika robotic arm with manipulating three distinct elements: a sliding door that can be slid open or closed, a cabinet door that can be open or closed and a knob that can rotate to control the stove burners, as shown in Fig 3. At each time step, the policy chooses a position and orientation for the end effector, or opens/closes the gripper. These three elements represent distinct types of motion, each of which require different control strategies. The

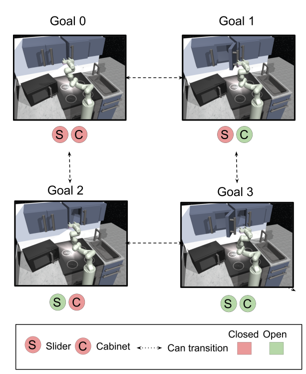

Fig. 4: Simulated tasks in kitchen. Tasks involve manipulating the kitchen cabinet and the sliding door to achieve various combinations of configurations.

[1]We overload the + operator here to denote an update to the density ρ when accounting for new states.

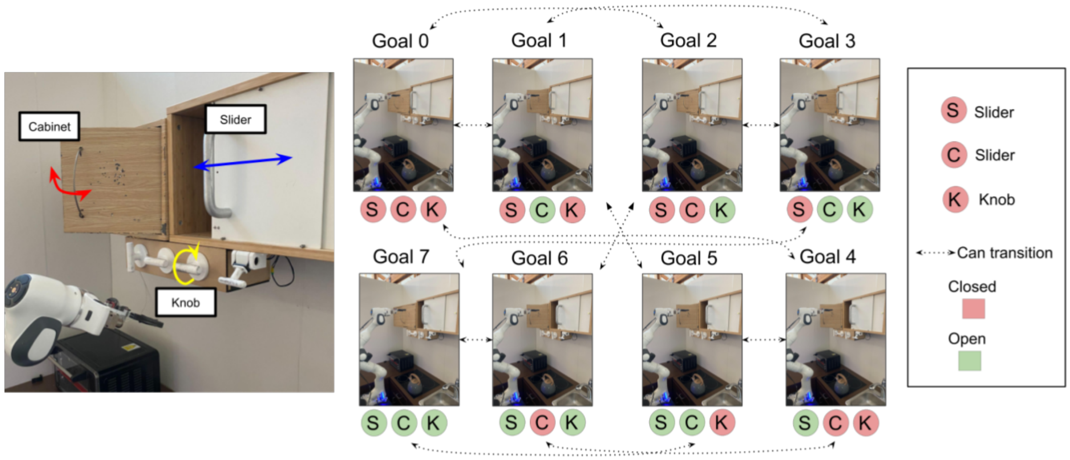

Fig. 3: Elements, tasks, and goals in the real-world kitchen environment. The agent is manipulating the cabinet, slider and knob to accomplish particular configurations as shown in goals 0 to 7. The dotted lines represent individual transitions, toggling one element at a time between its extreme positions. The goal of the agent is to learn a policy and a graph controller that is able to transition between goal states.

goal states of interest s_g^i are defined as combinations of the elements being opened or closed (or in the case of the knob, turned by 0 degrees or 90 degrees), resulting in $2^3 = 8$ goal states based on combinations of the elements being open or closed (Fig 3).

As described in Sections III, IV, the agent must practice reaching the goals of interest autonomously in the environment to master going from any combination of element configurations to any other. The agent is said to be in one of the goal states of interest if the current state of the elements in the scene are within ϵ distance of the goal.

Data collection. We make use of a teleoperation system to provide a continuous sequence of multi-task demonstrations. We collect "play-style" [4], [1] demonstrations, where different tasks are attempted successfully one after the other, indicating both how tasks are solved and how they may be sequenced. We make a simple change to the data collection procedure where the user indicates when a particular goal state of interest is completed before transitioning over to demonstrating a different goal. This allows the algorithm to easily determine the goals of interest as the transition points between these human-provided demonstrations. We provide around 500 demonstrations in the real world, requiring 2.5 hours of data collection time.

Simulation Environment. To provide thorough quantitative comparisons, we also evaluate our method on a simulated version of the above task, based on the MuJoCo kitchen manipulation environment described out by [1]. The purpose of this evaluation is to study the behavior of different components of the system in detail and more systematically run comparisons. In particular, in simulation we consider tasks with 2 elements (Fig 4): the cabinet and the slider. The goal states correspond to combinations of cabinet and slider being open and closed.

VI. Experimental Evaluation

In our experiments, we aim to answer the following questions: **(1)** Does DBAP allow agents to learn long horizon behaviors in the real world? **(2)** Does DBAP reduce the number of human resets required to learn long horizon behaviors, as compared to existing methods? **(3)** Can DBAP use prior data to bootstrap the learning of low-level policy execution and high level practicing behavior?

For real-world experiments, we compare DBAP to behavior cloning without finetuning ("Imitation" in Table I), offline RL with AWAC without finetuning ("Offline RL" in Table I) and the no pre-training baseline. The policies operate on a low-level state of object and endeffector positions, and they are represented with 3-layer MLPs of 256 units each (more details in supplementary material).

In simulation we compare with the following: **i) Non-pretrained low level, graph search high level:** This is a version of our algorithm, where the human-provided data is not used to bootstrap low level policies, but only the high level graph. This is related to ideas from [8], where multiple policies are learned from scratch, but unlike that work this baseline uses a learned high level graph. **ii) Pretrained low level, random high level controller:** This is a multi-task analogue to the perturbation controller [6]. The low-level policy is pre-trained from human demonstrations, but the high-level practicing chooses randomly which task to execute. **iii) Pretrained low level, BC task selection:** This is a version of the method by [1], modified for the minimal reset setting. Rather than using random task selection for practicing, tasks are sequenced using a high level policy trained with supervised learning on the collected data. **iv) Pretrained low level, reset controller:** This baseline is similar to a reset controller [2], [7], and alternates between commanding a random task and commanding a return to a single fixed start state during practicing. In this case, the high-level policy is a random controller. **v) Imitation low

	Success Rate	Path Length
Offline RL	0.83 ± 0.058	3.5 ± 0.17
DBAP (Ours)	$\mathbf{0.95 \pm 0.05}$	$\mathbf{3.37 \pm 0.3}$
Imitation	$0.62 \pm 0.$	4.3 ± 0.15
No Pretraining	0.0 ± 0.0	6.0 ± 0.0

TABLE I: Success rates and path lengths (in number of steps) in the real world for reaching long horizon goals.

level, imitation high level: This baseline involves training the both policies purely with imitation learning and running an offline evaluation the policy (as in [40]). **vi) Full relabeled relay policy learning (Gupta et al.):** It involves training the RPL algorithm [1] with fully relabeled goals, as opposed to just using labeled changepoints.

Evaluation metrics. We evaluate over multiple long horizon goal reaching manipulation tasks in the real world environment shown in Fig 3 and simulation shown in Fig 4. Our real-world environment has three elements (cabinet, knob, and slider) and two possible goal states per element (fully open or fully closed). Each long horizon task requires changing the original state of the environment to the inverted goal state. Success requires that a given task sequencer π^{hi} chain multiple low-level behaviors to accomplish the long-horizon goal. For example, given original state "cabinet open, knob open, and slider closed", the goal state would be "'cabinet closed, knob closed, and slider open". We construct 8 long horizon tasks in this manner.

A. Demonstration-Bootstrapped Practicing in Real World

We start by evaluating the ability of DBAP to learn multiple long horizon goal-reaching tasks in the real world. We trained the robot for over 25 hours as described in Section IV, only requiring a reset every 50 trajectories (in wall-clock time corresponding to once an hour only). This makes training much more practical in terms of human effort.

As seen from the evaluation performance of offline RL on long horizon success in Table I, DBAP starts off doing reasonably well during pre-training, achieving a 83% success rate, but improves significantly during autonomous practicing (with only very occasional resets) to around a 95% success rate, indicating the importance of online finetuning. The graph search automatically commands the low-level policy to reach each of the subgoals for the complete task. The learned behaviors are best appreciated by viewing the supplementary video.

In comparison, the no-pretraining, from-scratch baseline results in 0% success rate. This indicates the importance of being able to incorporate prior data, as is done in DBAP, to overcome hard exploration challenges inherent in long horizon manipulation. DBAP also significantly outperforms imitation learning in terms of long horizon goal-reaching performance. We compare the number of tasks commanded to accomplish the multi-step goals and observe the average path length is significantly lower for our method than baselines.

B. Demonstration-Bootstrapped Practicing in Simulation

Next, to study comparisons with baselines more systematically we evaluate the ability of DBAP to learn to accomplish multi-step goals in simulation as described in Section V, with a minimal amounts of automatically provided resets. In simulation we assume access to one human reset every 10 episodes, reducing the amount of human supervision by an order of magnitude as compared to standard RL. As shown in Fig 5 (left), our method successfully reaches a variety of multi-step goals and shows improvement over time with minimal human involvement (from 50% to 90% success rate). While other techniques are able to occasionally succeed, they take significantly longer to get to the goal, often resulting in roundabout paths to achieve the goal. This leads to the extended path length of trajectories in Fig 5 (middle). Training a high level policy via goal relabeling and BC (the "pretrained low-level, BC task selection" and "Full relabeled relay policy learning" baseline in Fig 5.) can inherit the biases of the data. In contrast, our graph-based search method is able to find and practice shorter paths.

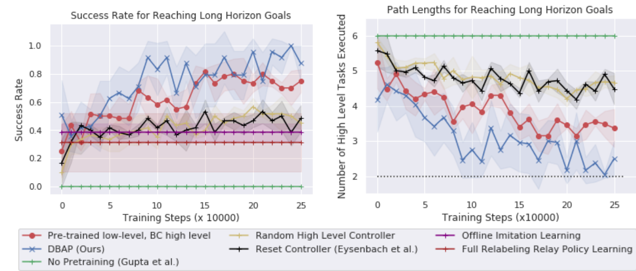

Fig. 5: Performance on the simulated environment in Fig 4 **(Left)** Success rate averaged across 3 seeds each on long horizon goals **(Right)** Average number of sub-goals commanded to reach a long horizon goal

VII. Discussion and Limitations

In this work, we described the design principles behind a system for learning behaviors autonomously using RL. We leverage human-provided data to bootstrap a multi-task RL system, using some tasks to provide resets for others. Our method combines offline RL at the low level and model-based graph search at the high level, where prior data is used to bootstrap low-level policies as well as facilitate practicing.

There are a number of interesting future directions we can pursue based on this work. One would be to obviate the need for state estimation, instead learning directly from visual inputs. Additionally, using these visual inputs to infer rewards directly will be important going forward. For building unconstrained reset free settings going forward, allowing for constantly expanding graphs would be interesting.

VIII. Acknowledgements

We would like to acknowledge the work by the operations team at Robotics at Google, especially work done by Thinh Nguyen and Gus Kouretas in building the kitchen environment used in this work, as well as Krista Reymann for her valuable help in managing project logistics. We also think Archit Sharma and Chelsea Finn for valuable discussion in the process of doing this work.

References

[1] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long horizon tasks via imitation and reinforcement learning. *Conference on Robot Learning (CoRL)*, 2019.

[2] Weiqiao Han, Sergey Levine, and Pieter Abbeel. Learning compound multi-step controllers under unknown dynamics. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6435–6442. IEEE, 2015.

[3] Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Gheshlaghi Azar, Bilal Piot, Nicolas Heess, Hado van Hasselt, Gregory Wayne, Satinder Singh, Doina Precup, and Rémi Munos. Hindsight credit assignment. In *NeurIPS*, 2019.

[4] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on Robot Learning*, pages 1113–1132. PMLR, 2020.

[5] John D Co-Reyes, Suvansh Sanjeev, Glen Berseth, Abhishek Gupta, and Sergey Levine. Ecological reinforcement learning. *arXiv preprint arXiv:2006.12478*, 2020.

[6] Henry Zhu, Justin Yu, Abhishek Gupta, Dhruv Shah, Kristian Hartikainen, Avi Singh, Vikash Kumar, and Sergey Levine. The ingredients of real-world robotic reinforcement learning. *arXiv preprint arXiv:2004.12570*, 2020.

[7] Benjamin Eysenbach, Shixiang Gu, Julian Ibarz, and Sergey Levine. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. *arXiv preprint arXiv:1711.06782*, 2017.

[8] Abhishek Gupta, Justin Yu, Tony Z Zhao, Vikash Kumar, Aaron Rovinsky, Kelvin Xu, Thomas Devlin, and Sergey Levine. Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention. *arXiv preprint arXiv:2104.11203*, 2021.

[9] Sehoon Ha, Peng Xu, Zhenyu Tan, Sergey Levine, and Jie Tan. Learning to walk in the real world with minimal human effort. *arXiv preprint arXiv:2002.08550*, 2020.

[10] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279, 2013.

[11] Simon Manschitz, Jens Kober, Michael Gienger, and Jan Peters. Learning movement primitive attractor goals and sequential skills from kinesthetic demonstrations. *Robotics and Autonomous Systems*, 74:97–107, 2015.

[12] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[13] Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*, 2018.

[14] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Survey: Robot programming by demonstration. Technical report, Springrer, 2008.

[15] Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. GTI: learning to generalize across long-horizon tasks from human demonstrations. In Marc Toussaint, Antonio Bicchi, and Tucker Hermans, editors, *Robotics: Science and Systems XVI, Virtual Event / Corvalis, Oregon, USA, July 12-16, 2020*, 2020.

[16] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[17] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

[18] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *arXiv preprint arXiv:1906.00949*, 2019.

[19] Ajay Mandlekar, Fabio Ramos, Byron Boots, Silvio Savarese, Fei-Fei Li, Animesh Garg, and Dieter Fox. IRIS: implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. In *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*, pages 4414–4420. IEEE, 2020.

[20] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

[21] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[22] Ziyu Wang, Alexander Novikov, Konrad Żołna, Jost Tobias Springenberg, Scott Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. *arXiv preprint arXiv:2006.15134*, 2020.

[23] Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[24] Ilya Kostrikov, Jonathan Tompson, Rob Fergus, and Ofir Nachum. Offline reinforcement learning with fisher divergence critic regularization. *arXiv preprint arXiv:2103.08050*, 2021.

[25] Avi Singh, Albert Yu, Jonathan Yang, Jesse Zhang, Aviral Kumar, and Sergey Levine. Cog: Connecting new skills to past experience with offline reinforcement learning. *arXiv preprint arXiv:2010.14500*, 2020.

[26] Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. *arXiv preprint arXiv:2102.08363*, 2021.

[27] Tatsuya Matsushima, Hiroki Furuta, Yutaka Matsuo, Ofir Nachum, and Shixiang Gu. Deployment-efficient reinforcement learning via model-based offline optimization. *CoRR*, abs/2006.03647, 2020.

[28] Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In *NeurIPS*, 1992.

[29] Thomas G. Dietterich. The MAXQ method for hierarchical reinforcement learning. In *ICML*, 1998.

[30] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 1999.

[31] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, 2017.

[32] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *NeurIPS*, 2018.

[33] Andrew Levy, George Dimitri Konidaris, Robert Platt Jr., and Kate Saenko. Learning multi-level hierarchies with hindsight. In *ICLR*, 2019.

[34] Ben Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *NeurIPS*, 2019.

[35] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In *ICLR*, 2018.

[36] Abdul Rahman Kreidieh, Samyak Parajuli, Nathan Lichtle, Yiling You, Rayyan Nasr, and Alexandre M. Bayen. Inter-level cooperation in hierarchical reinforcement learning. *arXiv*, 2019.

[37] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discret. Event Dyn. Syst.*, 2003.

[38] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[39] Carlos E. Garcia, David M. Prett, and Manfred Morari. Model predictive control: Theory and practice - A survey. *Autom.*, 25(3):335–348, 1989.

[40] Hoang Minh Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé III. Hierarchical imitation and reinforcement learning. In Jennifer G. Dy and Andreas Krause, editors, *ICML*, 2018.