Effective Offline Robot Learning with Structured Task Graph

Yiwen Hou*, Jinming Ma*, Haoyuan Sun and Feng Wu

***Abstract*—Offline reinforcement learning (RL) has shown great potential in many robotic tasks, where doing trial-and-error with the environment is risky, costly, or time-consuming. However, it is still hard to succeed in long-horizon tasks especially when given suboptimal and multimodal offline datasets. Nevertheless, existing RL methods rarely consider the structured information in offline datasets, which are commonly found in many robotic tasks. To address these challenges, we propose a novel offline RL approach that combines the techniques of dataset augmentation and subtask relabeling. Specifically, we first extract the subtasks and build the task graph based on the structured information in offline datasets. We then use the task graph to sample and generate an augmented dataset, which is more suitable for offline RL learning. After that, we relabel the dataset according to the task graph and finally learn a subtask-conditioned policy to complete the task. By doing so, we decompose the task of reaching a long-horizon goal state into a sequence of easier subtasks. This is not only useful for handling the long-horizon problem, but also reduces the error introduced by the offline dataset. We conducted extensive experiments in both the D4RL benchmark dataset and real-world robot with complex manipulation tasks. The experimental results show that our method significantly advances the state-of-the-art baselines in most tasks, particularly in long-horizon manipulation tasks with limited human demonstrations.**

***Index Terms*—Reinforcement Learning, Learning from Demonstration.**

I. Introduction

In many robotic tasks, reinforcement learning (RL) has achieved great success in real-world scenarios [1], [2]. However, the learning paradigm of RL usually requires a large amount of interactions with the environment, which may be risky, costly, or time-consuming. For instance, in the kitchen task [3] as shown in Fig. 1, the robot needs to place several items in target locations. Obviously, interacting with the complex kitchen environment via trial-and-error is impractical and may cause damage to the robot or environment. Most recently, offline RL has been proposed and offers a promising approach for robotic applications. Compared to its online counterpart, it enables learning policies merely from previously collected datasets. Hence it is more effective for the RL robot to learn policy offline for complex tasks.

Manuscript received: September, 3, 2023; Revised November, 28, 2023; Accepted December, 26, 2023.

This paper was recommended for publication by Editor-in-Chief Tamim Asfour and Editor Aleksandra Faust upon evaluation of the Reviewers' comments.

This work was supported in part by the Major Research Plan of the National Natural Science Foundation of China under Grant 92048301 and in part by the Anhui Provincial Natural Science Foundation under Grant 2208085MF172.

All authors are with School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China. (E-mail: {houyiwen,jinmingm,sunhaoyuan}@mail.ustc.edu.cn, wufeng02@ustc.edu.cn). Feng Wu is the corresponding author.

* Equal contributions.

Digital Object Identifier (DOI): see top of this page.

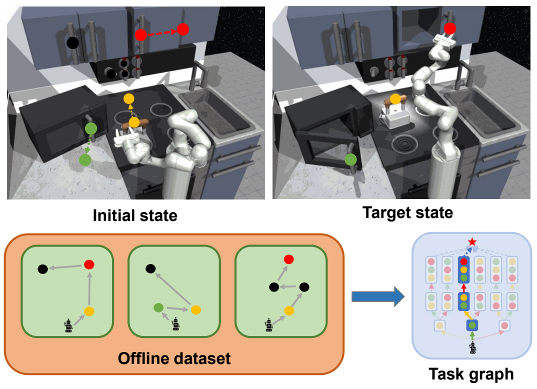


Fig. 1. **Top**: The target state is to have the microwave (green dots) and sliding cabinet door (red dots) open with the kettle (yellow dots) on the top burner. Black dots refer to other tasks, which is a distraction for reaching the target state. **Bottom**: From collected undirected offline data, we could build the task graph and sample the optimal subtask trajectories towards the target state (the bold path in the graph).

Unfortunately, recent efforts show that offline RL algorithms may fail to learn a reasonable policy due to the *error accumulation* during policy training [4], [5]. This is caused by the *distribution shift* between the current learned policy and the data distribution of the offline dataset, which leads to an overestimation of the Q-value in out-of-distribution (OOD) actions. Consequently, it will result in severe *one-step extrapolation errors* which accumulates via the Bellman backup operator, and as the task horizon increases, the rapidly increased *long-horizon error accumulation* will ultimately lead to the collapse of the entire policy learning. In complex robotic applications like the aforementioned kitchen task, the pre-collected data is sparse in the high-dimensional state and action spaces and the horizon is quite long, which is more likely to cause the *long-horizon error accumulation*.

Another key challenge in offline RL is the *suboptimality* and *multimodality* of the dataset due to the poor and diverse behavior policies for data collection. Specifically, most trajectories in the dataset may fail to or only partially complete the task. Assume that the kitchen task is to put the microwave, kettle, light, and cabinet to their target states respectively. Note that the items may be placed in different orders. In data collection, the robot may start with each item and try multiple solutions, which will make the dataset multimodal. Moreover, it is likely that the dataset is not specifically collected for the target task. Therefore, it is expected to be suboptimal and noisy in practice.

To date, researchers have studied several techniques to address the aforementioned challenges in offline RL based on constrained [4], [6]–[8], conservative [9]–[11] or in-sample [12], [13] methods. Another series of methods avoid the OOD issue based on imitation learning (IL) methods [14]–[16], which heavily depend on the quantity and quality of expert demonstrations [6], [17]. However, those methods are not very effective because they have not explored certain *structured information* concealed within the datasets. These structured information is often readily available in robotic applications. For example, in the kitchen task, placing different items has structural features between subtasks. In navigation tasks, there is usually a structure of the road network. Such structured information can help us break down tasks and organize datasets in a more structured manner. As shown later in our experiments, this is beneficial for tackling long-horizon complex tasks with suboptimal data.

In this paper, we propose a novel offline RL approach to utilize the structured information and address the key challenges of long-horizon tasks with suboptimal datasets. The basic idea is to decompose the long-horizon difficult task into short and simple subtasks by utilizing the structured information in the datasets. To this end, we first introduce the concept of *task graph* and build it from the offline dataset, as shown in Fig. 1. This task graph is useful to find the optimal subtask sequence from each subtask to the complete task. Based on it, we then sample complete trajectories to form the augmented dataset. Intuitively, the augmented dataset has better coverage of the path toward the overall task, which is beneficial to offline policy learning. After that, we relabel the dataset according to the task graph, which will decompose the long-horizon complex tasks and alleviate the *long-horizon error accumulation*. Given the relabeled data, we apply existing offline RL methods (e.g., BCQ [6], IQL [12]) to train a policy for completing the whole task. We first conducted several experiments in the common offline RL benchmark — the D4RL dataset [18] and tested on manipulation (i.e., Adroit and FrankaKitchen) tasks. We further validate the effectiveness of our method in a real-world manipulation task. Experimental results show that our method achieved substantially better performance than the state-of-the-art, in the difficult long-horizon manipulation tasks with suboptimal and multimodal datasets.

II. Problem Statement

We model the problem as a Markov Decision Process (MDP): $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \rho_0, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space; $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the environment dynamics; $\rho_0 \in \Delta(\mathcal{S})$ is the distribution of the initial states; $r(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function; $\gamma \in (0, 1]$ is the discount factor. The goal of RL is to learn a policy $\pi(a|s) : \mathcal{S} \to \Delta(\mathcal{A})$ that maximizes the cumulative discounted returns $J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$.

Here, we consider that the robot can only learn its policy from a pre-collected offline dataset $\mathcal{D} = \{\tau_i\}_{i=1}^{N}$, where each trajectory $\tau = (s_0, a_0, s_1, a_1, \dots)$ with $s_0 \sim \rho_0$. It is worth noting that the reward r is not required in our setting, which may not be readily available in real-world logged datasets.

As aforementioned, our goal is to solve the offline RL problem for domains with long horizon tasks using datasets collected by suboptimal behavior policies.

III. Main Method

We propose a novel approach for offline RL based on the structured information in the dataset. As shown in Fig. 2, we first build a task graph using the trajectories in the offline dataset. Then we sample the new complete trajectories from the task graph and form the augmented dataset. Finally, we train a policy using the augmented dataset with subtask relabeling. In what follows, we will give more details about the major procedures of our method.

A. Building Task Graph from Offline Dataset

We first introduce the concept of task graph and then propose the process of building it from the offline dataset. The basic idea is that we want to decompose complex long-horizon tasks into a sequence of easier subtasks and build links between the subtasks according to the offline dataset. After building the task graph, we can find the optimal subtask sequence via subtasks to complete the final task.

Specifically, given the subtasks $\mathcal{G}_{sub} = \{g_i \mid g_i \in \mathcal{S}, i = 1, 2, \dots, n_g\}$ where n_g is the number of subtasks, the task graph is a weighted and directed graph: $G = \{V, E\}$ with a weight function $w : E \to \mathbb{R}$. As shown in Fig. 2, each node $v \in V$ corresponds to a sequence of subtask states $\left[g^0, g^1, g^2, \dots, g^k\right]$, where superscript k represents the k-th task to complete, $g^0 = g_0 = s_0$ is the initial state with no subtask completed, $g^1, g^2, \dots, g^k \in \mathcal{G}_{sub}$ and $\forall g^m, g^n \in \left[g^1, g^2, \dots, g^k\right], g^m \neq g^n$, representing the state of accomplishing the subtasks $\left[g^1, g^2, \dots, g^k\right]$ in order. For $\forall v_i, v_j \in V$, the directed edge e_{ij} exist if and only if $v_i[-1] = v_j[-2]$, meaning the last subtask in v_i is equal to the second last one in v_j. For example, as shown in Fig. 2, the root represents the node $[g_0]$, the directed edge from $[g_0]$ to $[g_0, g_2]$ means to complete the subtask g_2 from the initial state g_0, Similarly, the directed edge from $[g_0, g_2, g_3]$ to $[g_0, g_2, g_3, g_1]$ means to complete the subtask g_1 from the subtask g_3. We additionally add a virtual success node v_{succ} (star in Fig. 2), and link the nodes that complete all the subtasks to v_{succ} with a virtual edge (blue dashed line). The weight $w(e_{ij})$ from node v_i to v_j represents the estimate of the distance of edge e_{ij} from the subtask g_i to g_j, where $g_i = v_i[-1]$ and $g_j = v_j[-1]$.

When there are multiple paths from the initial node $[g_0]$ to the virtual success node v_{succ}, which means that multiple subtask sequences can complete the whole task, it is important to find the most suitable path for offline policy learning. With the task graph, we can find the optimal subtask sequence. Note that the optimal subtask sequence needs to consider both the travel step (i.e., l_{ij}) and the data count (i.e., N_{ij}) between subtasks, which is critical for offline settings. If the amount of data is not taken into account, the optimal subtask sequence may lie in some accidental short paths, which is not

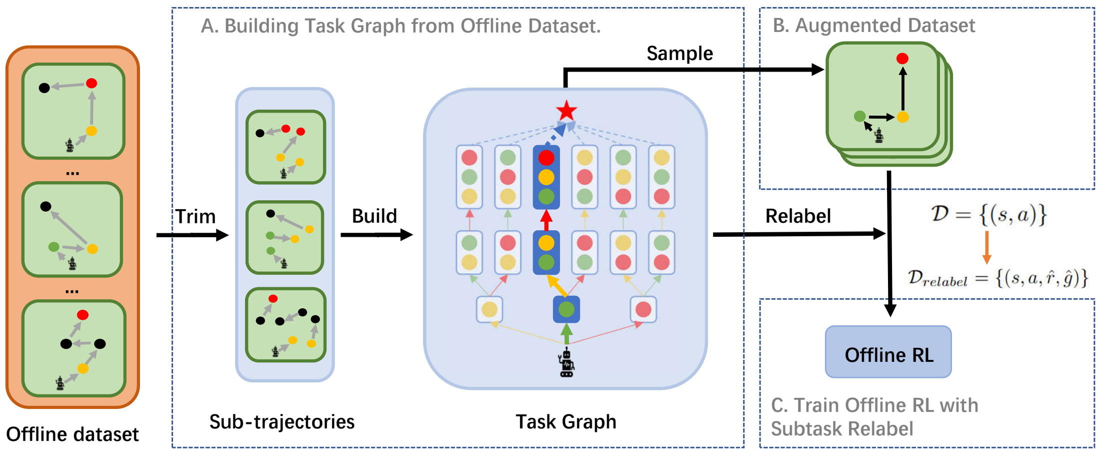

A. Building Task Graph from Offline Dataset.
B. Augmented Dataset
Sample
Trim
Build
Relabel
$\mathcal{D} = \{(s, a)\}$
$\mathcal{D}_{relabel} = \{(s, a, \hat{r}, \hat{g})\}$
Offline RL
C. Train Offline RL with Subtask Relabel
Offline dataset
Sub-trajectories
Task Graph

Fig. 2. Overview of offline robot task learning with Structured Task Graph from datasets. Each dot represents a subtask. The arrows in dataset mean the trajectories from one subtask to another, while the arrows in the Task Graph refer to the edges between the nodes (the subtask sequence).

conducive to offline policy learning due to the scarcity of data. If the travel step is not considered, learning policies between subtasks with longer steps will be more likely to suffer from long-horizon error accumulation. Thus, to make a trade-off between l_{ij} and N_{ij}, we define the weight $w(e_{ij})$ as

$$w(e_{ij}) = d(g_i, g_j) = l_{ij} + \alpha \frac{1}{N_{ij}} \tag{1}$$

where l_{ij} refers to the average trajectory time-step from g_i to g_j, N_{ij} refers to the number of trajectories from g_i to g_j that exist in the dataset, and α is the coefficient to balance the optimality and reachability. If there is no trajectory from the subtask g_i to g_j, the distance is considered infinite. We assign $w(e_{ij}) = 0$ for the virtual edges to v_{succ}.

Now, we can build the task graph based on the trajectories in the datasets as shown in Algorithm 1.

1) Detecting Subtasks from Offline Dataset: To build the task graph, we need to get a subtask set $\mathcal{G}_{sub}$, which is necessary to complete the whole task. We use *feature detector* $\Psi : s \to \mathcal{P}(F)$ to represent the abstract feature of the state s, which is commonly used to detect features in robot application [19]. Here, F is a set of propositions and $\mathcal{P}(F)$ is the power set of F. In real-world tasks, these detectors can be acquired through manually crafted rules based on sensory input (e.g., RGB-D cameras, force sensors) or by training CNNs to recognize the pose of specific objects relevant to the task. Note that we presume that the *feature detector* adequately decomposes the task, which is necessary for the construction of the task graph. With the feature detector Ψ representing the abstract feature of the state, an event occurs when the difference set of $p = \Psi(s_t) - \Psi(s_{t-1}) \neq \emptyset$, which means that a new proposition is true. We could utilize the event to trim the trajectories in the datasets, and find the subtask set $\mathcal{G}_{sub}$, each subtask $g \in \mathcal{G}_{sub}$ could be computed by the mean of the s_t when the corresponding event occurs.

Example 3.1: In the Kitchen task, the final task is to put the microwave, kettle, and cabinet in their target states respectively. So the proposition set F = {microwave is open, the kettle is at the target place, the cabinet is open}, we denote this as $F = \{m, k, c\}$ for brevity, and $\mathcal{P}(F) = \{\{\}, \{m\}, \{k\}, \{c\}, \{m, k\}, \{m, c\}, \{k, c\}, \{m, k, c\}\}$. The positions of the three items can be obtained through the RGB-D cameras, and the feature detector Ψ can be directly specified as whether the state of the object reaches the target state in such a case. As for the events, take the kettle subtask as an example, if $\Psi(s_t) = \{m\}$ and $\Psi(s_{t+1}) = \{m, k\}$, which means the kettle is successfully placed in the desired place at $t+1$, we put s_{t+1} in a set $\mathcal{G}_k$. Finally, we compute the mean of the states in $\mathcal{G}_k$ (i.e., $mean(\mathcal{G}_k)$) as the representation of the kettle subtask.

2) Generating Task Graph with Subtasks: With the event and corresponding subtask, we trim all trajectories $\mathcal{T}$ in the dataset into sub-trajectories $\mathcal{T}^{initial} = \{\mathcal{T}^{s_0 \to i}\}$ and $\mathcal{T}^{sub} = \{\mathcal{T}^{i \to j}\}$, where the initial state $s_0 \sim \rho_0$, $i, j \in \{g_1, g_2, \ldots, g_{n_g}\}$ and $i \neq j$. Here, $\mathcal{T}^{i \to j}$ represents the sub-trajectories that reach the subtask g_j from the subtask g_i and do not pass through other subtasks. So $\mathcal{T}^{initial}$ contains all sub-trajectories from the initial state to one of the subtasks, and $\mathcal{T}^{sub}$ contains all sub-trajectories from one subtask to another. Finally, we build the subtask graph G with nodes corresponding to subtask sequence state and edge weights estimated by Eq. 1.

Example 3.2: In the Kitchen task, suppose a trajectory completes (k, m, c) in sequence at time (t_1, t_2, t_3). We put $\tau[0 : t_1]$ into $\mathcal{T}^{s_0 \to k}$, $\tau[t_1 : t_2]$ into $\mathcal{T}^{k \to m}$, and $\tau[t_2 : t_3]$ into $\mathcal{T}^{m \to c}$. What's more, for the built task graph in the Kitchen example, the node $[g_0, g_m, g_c]$ refers to an abstract state that completes g_m and g_c in order from initial state g_0. Regarding edge weights, the edge weight from node $v_1 = [g_0, g_m]$ to node $v_2 = [g_0, g_m, g_c]$ is $w(e_{12}) = l_{mc} + \alpha \frac{1}{N_{mc}}$, where l_{mc} and N_{mc} are statistics on $\mathcal{T}^{m \to c}$.

B. Augmenting Dataset via Sampling Task Graph

As aforementioned, the offline dataset may contain many sub-optimal trajectories. Here, we augment the dataset by sampling optimal trajectories from the task graph. Intuitively, those trajectory samples can help policy learning avoid meaningless explorations and make it more stable and faster to converge.

Algorithm 1 Task Graph Generation

```
Input: feature detector $\Psi$, offline dataset $\mathcal{D}$
 1: Initialize $\mathcal{T}^{initial} \leftarrow \emptyset$
 2: Initialize $\mathcal{T}^{p_i \to p_j} \leftarrow \emptyset, \forall p_i, p_j \in F, p_i \neq p_j$
 3: Initialize $\mathcal{G}_p \leftarrow \emptyset, \forall p \in F$
 4: for $\mathcal{T}$ in $\mathcal{D}$ do        ▷ Trim offline dataset $\mathcal{D}$ based on $\Psi$
 5:     $ind\_start = 0$                      ▷ Initialize start index
 6:     for $t = 1, \dots, T$ do
 7:         if $p = \Psi(s_t) - \Psi(s_{t-1}) \neq \emptyset$ then
 8:             if $ind\_start = 0$ then
 9:                 $\mathcal{T}^{initial} \leftarrow \mathcal{T}^{initial} \cup \mathcal{T}[ind\_start, t]$
10:             else
11:                 $\mathcal{T}^{p' \to p} \leftarrow \mathcal{T}^{p' \to p} \cup \mathcal{T}[ind\_start, t]$
12:             $ind\_start = t, p' = p, \mathcal{G}_p \leftarrow \mathcal{G}_p \cup s_t$
13: $\mathcal{G}_{sub} \leftarrow \emptyset$
14: for $p$ in $F$ do                          ▷ Subtask computation
15:     $\mathcal{G}_{sub} \leftarrow \mathcal{G}_{sub} \cup mean(\mathcal{G}_p)$
16: $\mathcal{T}_{sub} \leftarrow \bigcup \mathcal{T}^{g_i \to g_j} \forall g_i, g_j \in \mathcal{G}_{sub}, g_i \neq g_j$
17: $\mathcal{D}_{trim} \leftarrow \mathcal{T}_{initial} \cup \mathcal{T}_{sub}$
18: Build the task graph $G$ with the node corresponding to
    subtask sequence state and the edge weight estimated
    following Eq. 1
Output: task graph $G = \{V, E\}$, trimmed dataset $\mathcal{D}_{trim}$
```

Given the task graph G, we can find the optimal task sequence from the initial node $[g_0]$ to the virtual success node v_{succ}. Without loss of generality, we assume that the optimal sequence of subtasks is $[g_0, g_1, g_2, ..., g_n]$. To get the sub-trajectory from $g_i \to g_{i+1}$, we sample $\tau_{i \to i+1}$ and concatenate those sub-trajectories into a complete trajectory, i.e., $\tau = concatenate(\tau_{g_0 \to g_1}, \tau_{g_1 \to g_2}, ..., \tau_{g_{n-1} \to g_n})$. Then this sampled trajectory τ is put back in the data set.

Note that the trajectories in original datasets may be generated by multiple behavior policies, which exhibit high diversity in how sub-task instances are solved. For example, the robotic arm can grasp different positions of the cabinet door and then open it. Given the optimal subtask sequence in the task graph, there are still many possible sub-trajectory from g_i to g_j in $\mathcal{T}^{i \to j}$. In order to generate better trajectories, we use the weighted sampling technique. Specifically, when sampling $\tau_{i \to j} \sim p(\tau^k_{i \to j})$, the shorter sub-trajectory will be given a higher sampling probability as:

$$p(\tau^k_{i \to j}) = \frac{\exp(\gamma^{d^k_{i \to j}})}{\sum_{n=1}^{N} \exp(\gamma^{d^n_{i \to j}})}, \quad \forall \tau^k_{i \to j} \in \mathcal{T}^{i \to j} \tag{2}$$

where $N = |\mathcal{T}^{i \to j}|$ is the number of sub-trajectories in $\mathcal{T}^{i \to j}$, $k = 0, 1, \dots, N-1$ is the kth trajectory in $\mathcal{T}^{i \to j}$, $d^k_{i \to j}$ represents the distance of $\tau^k_{i \to j}$ defined in Eq. 1, and γ is the discount factor. By sampling the trajectory repeatedly, we obtain the augmented dataset, containing optimal and unimodal distributed trajectories.

C. Learning Policy with Subtask Relabeled Dataset

To alleviate the severe extrapolation error problem in long horizon tasks, we use the subtask relabeling based on the task graph, which provides a better relabeling for the offline dataset.

Algorithm 2 Dataset Augmentation and Subtask Relabeling

```
Input: Task graph $G = \{V, E\}$, Trimed data $\mathcal{D}_{trim}$, Aug-
    ment percent $K$
 1: Initialize $\mathcal{D}_{Aug} \leftarrow \mathcal{D}_{trim}, \mathcal{D}_{relabel} \leftarrow \emptyset$
 2: Sample $s_0 \sim \rho_0$, find optimal subtask sequence
    $\{g_1, g_2, ..., g_n\}$ from initial state $s_0$ to final goal based
    on task graph $G$
 3: repeat                                   ▷ Dataset Augmentation
 4:     for i = 0, ..., n-1 do
 5:         sample $\tau_{i \to i+1} \sim p(\tau^k_{i \to i+1})$ with Eq. 2
 6:         $\mathcal{D}_{Aug} \leftarrow \mathcal{D}_{Aug} \cup \tau_{i \to i+1}$
 7: until $size(\mathcal{D}_{Aug}) > K \times size(\mathcal{D}_{trim})$
 8: for $\tau_{i \to j}$ in $\mathcal{D}_{Aug}$ do          ▷ Subtask Relabeling
 9:     $\tau_{relabel} \leftarrow \emptyset$
10:     for transition $(s, a, r, g)$ in $\tau_{i \to j}$ do
11:         $\hat{g} = g_j, \hat{r} = \mathbb{I}(\phi(s) \in \mathcal{G}_{sub})$
12:         $\tau_{relabel} \leftarrow \tau_{relabel} \cup (s, a, \hat{r}, \hat{g})$
13:     $\mathcal{D}_{relabel} \leftarrow \mathcal{D}_{relabel} \cup \tau_{relabel}$
14: Train the policy $\pi$ on $\mathcal{D}_{relabel}$
Output: the final policy $\pi$
```

The insight behind is that extrapolation error will accumulate rapidly as the horizon of the bellman backup increases [5], [20], [21], by decomposing the long-horizon task into the subtasks, the horizon of the bellman backup will be short enough for the offline RL algorithm to fix the extrapolation error. Specifically, the trajectories in the augmented datasets $\mathcal{D}_{Aug}$ are relabeled with the subtask to form a relabeled dataset: $\mathcal{D}_{relabel} = \{(s, a, \hat{r}, \hat{g})\}$, where $\hat{g} \in \mathcal{G}_{sub}$ is the subtask and $\hat{r}$ is the new reward. The relabel procedure is based on the subtask set $\mathcal{G}_{sub}$ and the sub trajectories $\tau_{i \to j}$ split by the subtask in $\mathcal{D}_{Aug}$. For each transition (s, a, r, g) in $\tau_{i \to j}$, the task will be relabeled as g_j, well the new reward is specified as $\hat{r_t} = 1$ when $\phi(s_t) \in \mathcal{G}_{sub}$, otherwise $\hat{r_t} = 0$.

To learn a policy in the offline manner, we extend BCQ [6] by simply concatenating states s and desired subtasks g as input states. Note that we choose BCQ for its simplicity and effectiveness though any offline RL algorithm can be used. The policy conditioned on subtasks is trained as follows:

$$\begin{aligned} \pi(s) &= \arg\max\nolimits_{a_j} Q\left(s, a_j + \xi\left(s, a_j, g, \Phi\right), g\right), \\ \text{s.t.} \quad & a_j \sim G_w(s, g) \end{aligned} \tag{3}$$

where $\xi(s, a_j, g, \Phi)$ is the perturbation model and $G_w(s, g)$ is the generative model. The Q-network $Q_\theta(s, a, g)$ is trained with the loss function:

$$\begin{aligned} \mathcal{L} &= [Q_\theta(s, a, g) - Q_{target}]^2, \text{where} \\ Q_{target} &= r + \gamma \max_{a' \sim G_w(s', g)} Q_{\theta^-}(s', a', g) \end{aligned} \tag{4}$$

The overall algorithm is summarized in Algorithm 2.

IV. Experiments

We first evaluate our method on two complex manipulation robotic tasks (i.e., Adroit and FrankaKitchen) with different offline datasets. Additionally, we conduct ablation studies to better understand the effectiveness of our key techniques.

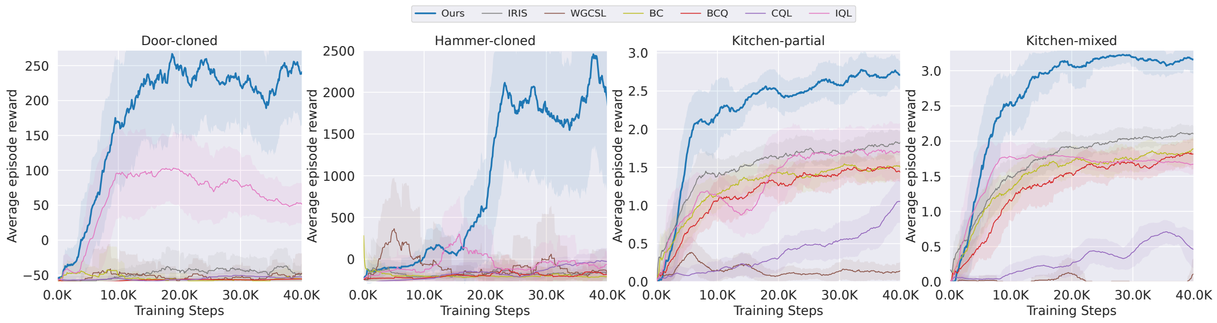


Fig. 3. **Adroit and FrankaKitchen:** learning curve our method and the baselines. Results are averaged over 5 random seeds, and the shadow is the standard deviation. Here, we focus on cloned dataset. In practice, it is more realistic to learn from human demonstrations and cloned dataset.

Finally, we evaluate our method on a challenging real-world manipulation task.

A. Manipulation Experiments

1) Experiment Settings: The Adroit domain involves controlling a 24-DoF simulated hand tasked with hammering a nail or opening a door. In the tasks, we tested in the following datasets: i) *human*: 25 sub-optimal human demonstration trajectories, ii) *cloned*: trajectories collected by an imitation policy trained with the demonstrations + demonstration trajectories, and iii) *expert*: a large number of trajectories from a fine-tuned RL policy. In the *human* and *cloned* datasets, there are many suboptimal trajectories and very narrow data distributions, which is difficult to learn an effective policy.

The FrankaKitchen domain involves controlling a 9-DoF Franka robot in a kitchen environment to manipulate multiple objects (e.g., microwave, kettle, etc.). We test on 3 datasets of human demonstrations: i) *complete*: 19 near-optimal demonstration trajectories completing all the desired tasks in order, ii) *partial*: only a subset of the desired tasks in each trajectory is completed, and the dataset is mixed with lots of trajectories not necessarily related to the desired tasks, and iii) *mixed*: no trajectories which solve the task completely, only a proper subset of the desired tasks in each trajectory is completed. These tasks are really challenging in terms of both the dataset composition and high dimensionality.

2) Baseline Methods: We compare our method with the leading offline RL algorithms: 1) **BC** [22]: a popular imitation learning method; 2) **BCQ** [6]: BCQ aims to perform Q-learning with batch-constrain; 3) **CQL** [10]: CQL learns a lower-bound or conservative Q-function for OOD actions; 4) **IQL** [12]: IQL avoids querying values of unseen actions; 5) **WGCSL** [15]: the weighted goal-conditioned supervised learning, re-weighting the trajectories with advantage value; 6) **IRIS** [16]: a hierarchical RL, with a high-level offline RL goal planner and a low-level imitation learning policy.

3) Results: As shown in Fig. 3, our method outperforms the other baselines in the speed of convergence, the stability of learning, and the solution quality. This confirms that our method can effectively solve long-horizon tasks with suboptimal and multimodal datasets.

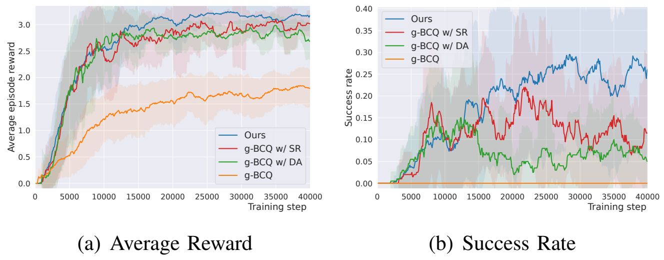


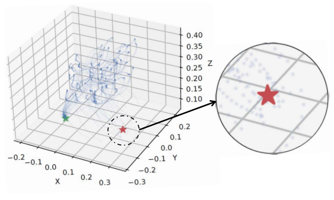
Fig. 4. Experiments for different modules in Kitchen-mixed dataset.

In Table I, we report the evaluation results on both of the tasks. In tasks with sufficient and high-quality datasets, most baselines can achieve good results, such as kitchen-complete. With high-quality datasets, IQL and WGCSL perform well, since they utilize imitation learning with exponential advantage weight. But they failed in the tasks with the suboptimal dataset, i.e., kitchen-mixed and kitchen-partial. Compared to these methods, our method deals with long-horizon tasks by utilizing the task graph and achieved good performance. This is due to the subtask decomposing the long-horizon difficult task into some short-horizon simple subtasks, which makes the backup of Q-values faster and less likely to cause long-horizon error accumulation.

For more complex dexterous manipulation tasks, most baselines fail to learn policy, such as hammer-human and hammer-cloned. In off-policy methods, such as **BCQ** and **CQL**, the error accumulation is further exacerbated by the high-dimensional state-action space and limited dataset. While imitation learning methods, such as **IQL** and **BC**, encounter the challenge of compounded errors resulting from long-horizon tasks. **WGCSL** did not perform well with human or cloned datasets, since the inadequate coverage of the desired goal distribution and ultimately hindered performance on tasks requiring long-term planning. **IRIS** also faces the issue of inaccurate estimation of the value function for goal planner. From the results, we can see that our method substantially outperformed all the compared methods, especially in the more challenging dexterous manipulation tasks. This confirms that our method can effectively solve long-horizon tasks with suboptimal datasets.

TABLE I
AVERAGE SCORE OF ALL METHODS IN THE THREE DOMAINS WITH DIFFERENT TYPE OF THE DATASETS. (BEST VALUES ARE IN BOLD).

Domain	Dataset type	BCQ	BC	CQL	IQL	WGCSL	IRIS	Ours
Adroit	hammer-human	-233.13	27.01	300.2	23.2	41.35	64.42	**3122.68**
	door-human	-59.21	-1.61	234.3	72.4	7.78	31.70	**275.84**
	hammer-cloned	-117.47	-219.99	-13.15	-69.61	-134.59	-173.22	**2045.11**
	door-cloned	-21.22	-58.24	-51.40	58.28	-46.19	-49.64	**246.07**
	hammer-expert	16294.25	15286.51	15902.1	15753.0	16205.98	16283.52	**16336.14**
	door-expert	2804.60	3009.40	2926.8	2945.2	2990.44	3019.44	**3022.88**
FrankaKitchen	kitchen-complete	1.53	1.16	0.38	**2.5**	2.01	1.88	1.91
	kitchen-partial	1.44	1.43	1.23	1.68	0.16	1.82	**2.76**
	kitchen-mixed	1.76	1.90	0.14	1.59	0.32	2.08	**3.17**


(a) Offline Dataset

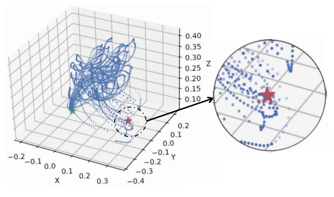

(b) Augmented Dataset

Fig. 5. Distribution of the original and augmented dataset in Door-human dataset. In the given visual representation, the start of the task is denoted by the green star while the end of the task is denoted by the red star.

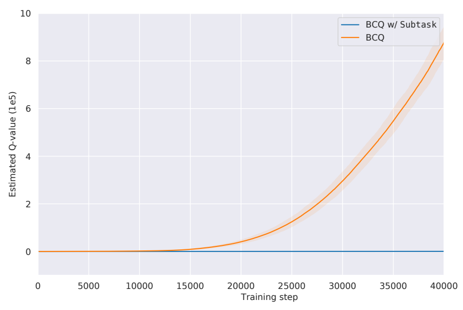


Fig. 6. Estimated Q-values in the Adroit Domain.

B. Ablation Experiments

We firstly study the usefulness of our *data augumentation* and *subtask relabeling* modules in Kitchen-mixed dataset. To this end, we trained g-BCQ with only *subtask relabeling*, g-BCQ with only *dataset augmentation*, and g-BCQ with both *subtask relabeling* and *dataset augment* (i.e., g-BCQ w/ SR, g-BCQ w/ DA, and Ours), where g-BCQ is the simple extension of BCQ with the concatenate of state and target task as input. As shown in Fig. 4, the *subtask relabeling* can improve the success rate, while the policy trained without it has a significant decrease in performance. As mentioned, the purpose of subtask relabeling is to decompose difficult tasks into multiple easy tasks to make policy learning easier. Meanwhile, the dataset augmentation stabilizes the training process, making the robot better understand how to successfully complete the task. All in all, both of the modules are effective for offline robot learning.

1) Data Augmentation: To better understand the role of our *data augmentation* module, we visualize the data distribution of the augmented data in the Adroit domain. As shown in Fig. 5, the original data distribution contains a limited number of trajectories that have successfully completed the task (reached the red star), whereas the augmented data distribution provides improved coverage of the path leading to the red star, which is beneficial for the training process.

TABLE II
AVERAGE SCORES OF OUR METHOD WITH DIFFERENT UNDERLYING ALGORITHMS. (ORIG. REFERS TO THE ORIGINAL VERSION)

	BCQ		BC		CQL		IQL	
	Orig.	**Ours**	**Orig.**	**Ours**	**Orig.**	**Ours**	**Orig.**	**Ours**
kitchen-complete	1.53	**1.91**	1.16	**2.10**	0.38	**0.63**	2.50	**2.58**
kitchen-partial	1.44	**2.76**	1.43	**2.61**	1.23	**1.63**	1.68	**2.64**
kitchen-mixed	1.76	**3.17**	1.90	**2.82**	0.14	**1.60**	1.59	**3.04**

2) Subtask Relabeling: We plot the estimation of the Q-value during a training epoch in the Door domain with the cloned dataset, which is narrowly distributed. From Fig. 6, the estimation of the BCQ goes to infinite, which is pathological and unrealistic. This phenomenon of offline training is caused by the long-horizon error accumulation. Although we use BCQ for handling OOD, it still easily occurs due to the high dimensionality of the state space and narrowly distributed dataset. However, when decomposing the task into subtasks, we reduce the possibility of iterative error exploitation [5] by shortening the horizon of the estimated Q-value in the Bellman backup.

3) Different underlying algorithms: We validate the adaptation of the proposed method to different underlying algorithms in the Kitchen domain. As shown in Table II, our method consistently outperforms all the original offline RL baselines in all datasets, demonstrating the generality of our methods to any downstream offline RL algorithms. This shows that by effectively utilizing the task graph, any downstream algorithms can be improved consistently.

4) Different weighting schemes: To further study the impact of different weighting schemes on the task graph, we conducted additional experiments on the Kitchen datasets. The results in Table III reveal that considering both the travel step (l_{ij}) and the data count (N_{ij}) results in the highest scores in all datasets. For datasets with homogeneous trajectories that have the same subtask completion sequence (i.e. Kitchen-complete), the weighting schemes do not affect. However, for

TABLE III
AVERAGE SCORES WITH DIFFERENT WEIGHTING SCHEMES

	Ours	Ours (only l_{ij})	Ours (only N_{ij})
kitchen-complete	**1.91**	**1.91**	**1.91**
kitchen-partial	**2.76**	1.64	**2.76**
kitchen-mixed*	**3.17**	**3.17**	2.03

the dataset with mixed multimodal data, our weighting metric is really effective. For the Kitchen-partial dataset, there exist some trajectories that occasionally complete certain tasks. If we only consider steps l_{ij}, the optimal subtask sequence lies in such fortuitous trajectories, and the limited amount of data leads to poor policy. As for the kitchen-mixed* dataset, we add some new trajectories to the original dataset by adding Gaussian noise to the state of some sub-trajectories. In such a dataset, the subtask sequence with the largest amount of data has longer steps. If we only consider N_{ij}, the optimal subtask sequence lies in such longer trajectories, resulting in a lower final score due to the longer horizon of the subtask.

C. Real-World Robotic Manipulation

We conducted an evaluation of our approach on a complex, long-horizon, real-world manipulation task. We use a 7-DoF Kinova Gen3 robotic arm equipped with a Robotiq gripper. As shown in Fig. 7, the task is to build a ladder with three different blocks. Initially, these blocks are spread out on the table and our goal is to align them to build the desired shape.

The state space is represented by a 13-dimensional vector, including end-effector position (3D), gripper opening state (1D), and position of three blocks (3D $\times$ 3). Here, the pose estimation of the blocks is obtained by Apritag [23] from the RGB image, which is captured from the RealSense depth camera D455. The action space is a 4-dimensional vector, responsible for dictating the velocity of the end-effector (3D) and the status of the gripper (1D, i.e., open or close). At each time-step, the reward corresponds to the number of blocks correctly positioned given the target configuration.

We utilized a rule-based policy written by hands to collect six different types of trajectories. One type of trajectories precisely completes our goal task — building the ladder. However, the other five types of trajectories either partially achieve the goal task or misplace some blocks, acting as potential distractions from the goal task. For each trajectory type, we gathered 20 trajectories, for a total of 120 trajectories. By doing so, the collected dataset contains multimodal trajectories with suboptimal and narrow data distribution, which is very challenging for offline RL.

We conduct offline training for all methods with the same datasets and then test the trained policies in the real-world robotic manipulation task. We train different instances of each method with different random seeds, with each instance performing 10 evaluation rollouts. The results of the evaluation are reported in Table IV. We report the Average return and Average task completion ratio of evaluation rollouts. The task

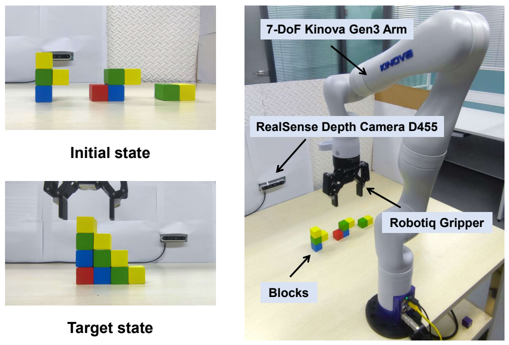


Fig. 7. Building blocks with Kinova Gen3 arm.

completion ratio represents the percentage of successfully placing blocks to build the ladder.

As shown, our method outperforms all baselines in terms of average return and task completion ratio. The **CQL** method encounters severe out-of-distribution errors, rendering it unable to pick up any blocks. Relying solely on dataset mimicking, the **BC** method misplaces some blocks due to the high multimodality of the dataset, which results in a low average return. Other baselines only partially complete the target task, which may be caused by the inaccurate value estimation of the long-horizon task. By utilizing the task graph, our method decomposes the target task into three subtasks and could successfully complete them in order.[1] The results confirm that our method with the task graph is also effective in real-world long-horizon robotic tasks.

V. RELATED WORK

A. Offline Reinforcement Learning

In offline RL, the agent needs to learn a policy from a fixed dataset collected by some behavior policies [6], [24]. As aforementioned, most offline RL methods are based on constrained, conservative, or in-sample methods. For policy-constraint methods, they constrain the learned policy to be close to the behavior policy. It can be implemented via an explicit action constraint [6], [7], using an implicit regularization [4], [8], or adding an uncertainty weight to the policy improvement objective [25], [26]. Conservative-based methods have also proposed to directly regularize the Q-function to produce lower-bound or conservative Q-function for OOD actions [9]–[11]. For example, CQL [10] adds a regularizer to penalize the Q-function of OOD actions and encourage the Q-function for state-action pairs in the dataset to be large. In-sample methods avoid querying values of unseen actions while still enabling multi-step dynamic programming [12], [13]. Diverging from prior work, we make further use of the structured information from offline data, which is crucial for robot learning.

B. Offline Goal-conditioned RL and Offline Hierarchical RL

Generally, our work is related to offline goal-conditioned RL. Among them, GCSL [27] relabels and simply does goal-conditioned behavior cloning on relabeled data. WGCSL [15]

[1]See the attached video for details: https://youtu.be/o-oFSCBaR24

TABLE IV
AVERAGE SCORES OF ALL METHODS IN REAL-WORLD TASKS (BEST VALUES ARE IN BOLD).

	BCQ	**BC**	**CQL**	**IQL**	**WGCSL**	**IRIS**	**Ours**
Average return	184.2	62.7	0.0	36.1	153.2	105.0	**342.6**
Average task completion ratio	33.3%	25.0%	0.0%	6.7%	27.8%	23.3%	**100.0%**

extends GCSL by introducing discounted relabeling weight, exponential advantage weight, and best-advantage weight to tackle the suboptimal and multimodal problem. Our work is also related to offline hierarchical RL. IRIS [16] uses a hierarchical framework to learn from offline datasets, with a high-level VAE and value function to select a subgoal and train a low-level goal-conditioned policy with imitation learning. HiGoC [28] takes advantages of model-based planning method as a high-level planner and train goal-conditioned polcily on expert data, which may not be readily accessible. In contrast, we make further use of the structured information from offline data to obtain better data distribution for offline RL. Moreover, we adopt subtask relabeling instead of future hindsight relabel to learn policy.

VI. Conclusion and Discussion

In this paper, we introduced an offline RL approach to solve long-horizon tasks with sub-optimal demonstrations. By first building the task graph from the offline datasets, we subsequently augment the dataset by sampling trajectories from the task graph, and relabel the dataset based on the task graph for policy learning. Our dataset augmentation technique improves the data distribution to solve the suboptimal and multimodal problem of the offline dataset, while the subtask relabeling decomposed the long-horizon tasks into simple tasks, making the learning process more stable and faster. Our experiments on the D4RL dataset and real-world manipulation task confirm the effectiveness of our method.

The major limitation of our proposed method is that when the dataset only contains homogeneous trajectories that have the same subtask completion sequence, our approach will not fully benefit. The more diverse the dataset, the more our method benefits. Another consideration is that if the task graph is quite large or complex, additional computation costs are required to find the optimal subtask sequence, while the optimality can be guaranteed. In the future, we plan to explore more effective techniques for learning subtasks in the high-dimensional space from the offline datasets and test our method on more challenging robotic tasks.

References

[1] S. Sinha, A. Mandlekar, and A. Garg, "S4rl: Surprisingly simple self-supervision for offline reinforcement learning in robotics," in *Conference on Robot Learning*. PMLR, 2022, pp. 907–917.

[2] A. Kumar, A. Singh, S. Tian, C. Finn, and S. Levine, "A workflow for offline model-free robotic reinforcement learning," *arXiv preprint arXiv:2109.10813*, 2021.

[3] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning," *arXiv preprint arXiv:1910.11956*, 2019.

[4] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine, "Stabilizing off-policy q-learning via bootstrapping error reduction," 2019.

[5] D. Brandfonbrener, W. Whitney, R. Ranganath, and J. Bruna, "Offline rl without off-policy evaluation," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4933–4946, 2021.

[6] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International conference on machine learning*. PMLR, 2019, pp. 2052–2062.

[7] S. Fujimoto and S. S. Gu, "A minimalist approach to offline reinforcement learning," 2021.

[8] Y. Wu, G. Tucker, and O. Nachum, "Behavior regularized offline reinforcement learning," *arXiv preprint arXiv:1911.11361*, 2019.

[9] T. Yu, A. Kumar, R. Rafailov, A. Rajeswaran, S. Levine, and C. Finn, "Combo: Conservative offline model-based policy optimization," 2021.

[10] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," 2020, pp. 1179–1191.

[11] Q. He, X. Hou, and Y. Liu, "Popo: Pessimistic offline policy optimization," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 4008–4012.

[12] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit q-learning," *arXiv preprint arXiv:2110.06169*, 2021.

[13] H. Xu, L. Jiang, J. Li, Z. Yang, Z. Wang, and X. Zhan, "Sparse q-learning: Offline reinforcement learning with implicit value regularization," in *3rd Offline RL Workshop: Offline RL as a"Launchpad"*.

[14] I. Kostrikov, O. Nachum, and J. Tompson, "Imitation learning via off-policy distribution matching," *arXiv preprint arXiv:1912.05032*, 2019.

[15] R. Yang, Y. Lu, W. Li, H. Sun, M. Fang, Y. Du, X. Li, L. Han, and C. Zhang, "Rethinking goal-conditioned supervised learning and its connection to offline rl," *arXiv preprint arXiv:2202.04478*, 2022.

[16] A. Mandlekar, F. Ramos, B. Boots, S. Savarese, L. Fei-Fei, A. Garg, and D. 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)*. IEEE, 2020, pp. 4414–4420.

[17] J. Li, X. Hu, H. Xu, J. Liu, X. Zhan, Q.-S. Jia, and Y.-Q. Zhang, "Mind the gap: Offline policy optimization for imperfect rewards," *arXiv preprint arXiv:2302.01667*, 2023.

[18] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," *arXiv preprint arXiv:2004.07219*, 2020.

[19] A. Camacho, J. Varley, A. Zeng, D. Jain, A. Iscen, and D. Kalashnikov, "Reward machines for vision-based robotic manipulation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 14 284–14 290.

[20] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," 2021, pp. 1273–1286.

[21] B. Eysenbach, R. R. Salakhutdinov, and S. Levine, "Search on the replay buffer: Bridging planning and reinforcement learning," 2019.

[22] M. Bain and C. Sammut, "A framework for behavioural cloning." in *Machine Intelligence 15*, 1995, pp. 103–129.

[23] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 3400–3407.

[24] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[25] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Y. Zou, S. Levine, C. Finn, and T. Ma, "Mopo: Model-based offline policy optimization," 2020, pp. 14 129–14 142.

[26] Y. Wu, S. Zhai, N. Srivastava, J. Susskind, J. Zhang, R. Salakhutdinov, and H. Goh, "Uncertainty weighted actor-critic for offline reinforcement learning," *arXiv preprint arXiv:2105.08140*, 2021.

[27] D. Ghosh, A. Gupta, A. Reddy, J. Fu, C. Devin, B. Eysenbach, and S. Levine, "Learning to reach goals via iterated supervised learning," *arXiv preprint arXiv:1912.06088*, 2019.

[28] J. Li, C. Tang, M. Tomizuka, and W. Zhan, "Hierarchical planning through goal-conditioned offline reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 216–10 223, 2022.