Final Project: Offline-to-Online Reinforcement Learning

Milestone report due April 13th
Final Report due May 13th

For the full final project outline, please refer to the Final Project Outline listed on the course website

AI Policy

> **You may not use AI assistants (chatbots, coding agents, etc.) to directly generate any material that will be graded. This includes both the code and PDF reports for both the homework assignments and the final project.**
>
> We are following the Stanford Generative AI Policy, which is that you may use AI assistants so long as you treat them as you would a human collaborator. For example, you can ask a friend to help explain a difficult concept, or ask them to look at a particularly obtuse error that you're stuck on. However, you can't hand them your laptop and tell them to start coding for you. Similarly, you can ask an AI assistant for help with difficult concepts or errors, but you cannot ask them to directly generate code or give them control of your terminal.

1 Overview

The final project is meant to be a less guided, more exploration-based, and longer version of the assignments you have done in the course so far. Less starter code will be provided, and more weight will be placed on your own exploration than on implementing prior methods. **This document focuses on Option II: Offline-to-online RL methods.**

1.1 Motivation

In the homework, you have and will explore several online and offline RL methods. However, each of these method classes has its advantages and disadvantages (see section 1.1).

In this project, you will explore ways to transition between these methods through **offline-to-online** RL methods. This class of methods initially trains a policy or value function on offline data, and then performs an online fine-tuning step. The key here is trading off the requirement for offline data with a budget for online interaction. You will perform your experiments in OGbench Park et al. (2025).

First, you will implement some naïve offline-to-online methods based on those from the homework, then you will explore methods that mitigate the problems that arise in offline-to-online methods, and finally, you will be tasked with trying to improve beyond these methods.

1.2 Tasks

For this project, you will be working with OG-bench (Park et al., 2025), which you will have also used in Homework 5. You are welcome to apply your method to any of the tasks, but you *must* show results for the methods you implement on:

	Offline RL	**Online RL**
Pros	• No interaction with env. required (safer, faster etc.). • Efficient - can reuse samples from previously collected datasets.	• Can explore the env. and expand the dataset beyond any pre-collected data. • Training data will be closer to the policy distribution (on-policy/off-policy online methods)
Cons	• Limited by the pre-collected dataset (poor coverage or bias is hard to recover from) • Our dataset distribution will not match the policy distribution at test-time.	• Requires lots of environment interaction (which can be unsafe - and very challenging for real-world RL!)

- `cube-single-play-singletask-task1-v0 (state-based)`: In this environment, the policy must place a cube at a target location. This will be our test environment for checking if our policies are working. We will provide guidelines for what to expect in this environment for each of the baselines you implement. Your implementations should be able to essentially saturate performance on this task.
- `cube-double-play-singletask-task1-v0 (state-based)`: Similar to `cube-single`, but with 2 cubes.
- `antsoccer-arena-navigate-singletask-task1-v0 (state-based)`: Similar to `antmaze`, the agent must learn to navigate, but in this case, it must also control a ball. In the arena setting, there is no maze to navigate, but the agent must move to a randomized goal with the ball.

1.3 [IMPORTANT] Restrictions

We will limit the number of offline and online training steps that can be used (for each experiment). In all your experiments, the maximum number of steps for each phase and the number of seeds are as follows:

1. **Offline steps:** 500,000
2. **Online steps:** 100,000
3. **Seeds:** 2 (for section 5)

You **must not** exceed this budget, or you will not be awarded points for that section.

Starter code is available here:

https://github.com/berkeleydeeprlcourse/homework_spring2026/tree/main/final_project_offline_online

We stay very close to the homework setup, so modal and autograding operate the same way as for homeworks.

2 Part 0: Layer Norm

Before doing anything else, we are going to implement something very simple but effective for all of our model architectures: layer norm. Layer norm has been shown to be very effective for improving training stability for RL methods Ball et al. (2023); Lyle et al. (2024). We will add this to our networks file `src/infrastructure/pytorch_util.py`. In the MLP definitions (see `TODOs`), add a layer norm between the linear layer and the activation. Your models should still train reasonably well without this, but this is a good practice!

3 Part I: Naïve Offline to Online RL

The most direct approach to offline-to-online RL is to take a policy trained on offline data, roll it out in the environment to collect online data, and then update the policy using this new data.

In this section, you will take two of our offline methods, which you will implement (or will have implemented) in **HW5**, SAC + BC, and FQL, and add a generic online finetuning step.

Concretely, you should combine the offline training loop from **HW5** with the online finetuning phase from **HW3**. We provide the same starter code as **HW5** provides for these agents and the same `run.py` file.

3.1 Algorithm overview

The same information is provided in **HW5** but is provided here for convenience.

SAC + BC

SAC+BC trains two Q functions $Q_1, Q_2 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a Gaussian policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, and a learnable entropy coefficient $\beta > 0$, where $\Delta(\mathcal{A})$ denotes the set of probability distributions over the action space $\mathcal{A}$. The Q functions are trained to minimize the following Bellman error:

$$\mathcal{L}(Q) = \sum_{i=1}^{2} \mathbb{E}_{(s,a,r,s') \sim \mathcal{D},\, a' \sim \pi(\cdot|s')} \left[(Q_i(s,a) - y)^2 \right], \tag{1}$$

$$\text{where } y = r + \frac{\gamma}{2} \sum_{j=1}^{2} \bar{Q}_j(s', a'), \tag{2}$$

where $\bar{Q}_1$ and $\bar{Q}_2$ are target networks for Q_1 and Q_2, respectively. The target networks are updated with Polyak averaging (*i.e.*, soft updates) after every gradient step. We omit the entropy term in the Bellman backup (deviating from the original SAC algorithm), as it generally works better in offline RL settings. We also use the average (instead of the minimum) of the two target Q values in the Bellman backup, which often works better in practice.

The policy is trained to minimize the following loss:

$$\mathcal{L}(\pi) = \mathbb{E}_{(s,a) \sim \mathcal{D},\, a^\pi \sim \pi(\cdot|s)} \left[-\frac{1}{2} \sum_{i=1}^{2} Q_i(s, a^\pi) + \alpha \cdot \frac{1}{|\mathcal{A}|} \|a - a^\pi\|_2^2 + \beta \cdot \log \pi(a^\pi \mid s) \right], \tag{3}$$

where α is a hyperparameter that controls the strength of the behavioral cloning term. Intuitively, the first term maximizes the Q values, the second term regularizes the policy to dataset actions, and the third term maximizes the entropy of the policy. The main difference from the original SAC actor loss is the addition of the behavioral cloning term (the second term). As in SAC, the policy is trained via the reparameterization trick.

Finally, the (learnable) entropy coefficient β is trained via dual gradient descent to match a target entropy $\bar{\mathcal{H}} \in \mathbb{R}$, by minimizing the following loss:

$$\mathcal{L}(\beta) = \mathbb{E}_{s \sim \mathcal{D},\, a^\pi \sim \pi(\cdot|s)} \left[\beta \cdot \left(-\log \pi(a^\pi \mid s) - \bar{\mathcal{H}} \right) \right]. \tag{4}$$

In practice, we typically set the target entropy to $\bar{\mathcal{H}} = -|\mathcal{A}|/2$, where $|\mathcal{A}|$ is the action dimension. If you're not familiar with this automatic entropy tuning technique (or dual gradient descent in general), we recommend reading Section 6 of the paper by Haarnoja et al. (2018b). In this assignment, we don't ask you to implement automatic entropy tuning, but provide an implementation of it in the starter code.

The most important hyperparameter in SAC+BC (and any other behavior-regularized methods in general) is the BC coefficient α. The performance is generally *highly* sensitive to the choice of α, and it needs to be tuned carefully for each task.

FQL

FQL trains two Q functions $Q_1, Q_2 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a *behavioral* flow policy defined by a time-dependent vector field $v : \mathcal{S} \times \mathcal{A} \times [0,1] \to \mathcal{A}$, and a *one-step* policy defined by a standard feedforward network $\pi_\omega : \mathcal{S} \times \mathbb{R}^{|\mathcal{A}|} \to \mathbb{R}^{|\mathcal{A}|}$, where $\mathcal{A} = \mathbb{R}^{|\mathcal{A}|}$ is the action space. As before, we denote the behavioral flow policy induced by v as $\pi_v : \mathcal{S} \times \mathbb{R}^{|\mathcal{A}|} \to \mathbb{R}^{|\mathcal{A}|}$.

The behavioral flow policy is trained with the standard flow-matching loss for behavioral cloning:

$$\mathcal{L}(v) = \mathbb{E}_{\substack{(s,a)\sim\mathcal{D}, \\ z\sim\mathcal{N}(0,I_{|\mathcal{A}|}), \\ t\sim\mathrm{Unif}([0,1])}} \left[\frac{1}{|\mathcal{A}|} \left\| v(s, \tilde{a}, t) - (a - z) \right\|_2^2 \right], \tag{5}$$

$$\text{where } \tilde{a} = (1-t)z + ta. \tag{6}$$

The Q functions are trained with the standard Bellman loss:

$$\mathcal{L}(Q) = \sum_{i=1}^{2} \mathbb{E}_{(s,a,r,s')\sim\mathcal{D},\, z\sim\mathcal{N}(0,I_{|\mathcal{A}|})} \left[(Q_i(s,a) - y)^2 \right], \tag{7}$$

$$\text{where } y = r + \frac{\gamma}{2} \sum_{j=1}^{2} \bar{Q}_j(s', \pi_\omega(s', z)), \tag{8}$$

where $\bar{Q}_1$ and $\bar{Q}_2$ are target networks for Q_1 and Q_2, respectively. Note that the policy used in the Bellman backup is the one-step policy π_ω, not the behavioral flow policy π_v.

Finally, the one-step policy is trained to minimize the following loss:

$$\mathcal{L}(\pi_\omega) = \mathbb{E}_{\substack{(s,a)\sim\mathcal{D}, \\ z\sim\mathcal{N}(0,I_{|\mathcal{A}|})}} \left[-\frac{1}{2} \sum_{i=1}^{2} Q_i(s, \pi_\omega(s,z)) + \frac{\alpha}{|\mathcal{A}|} \left\| \pi_\omega(s,z) - \pi_v(s,z) \right\|_2^2 \right], \tag{9}$$

where α is a hyperparameter that controls the strength of the distillation term. Here, recall that π_ω is a standard feedforward network, and π_v is an ODE integration of the vector field v from 0 to 1. Gradients do not backpropagate through π_v in the distillation term. Intuitively, FQL's one-step policy loss above interpolates between Q maximization (the first term) and distillation from the behavioral flow policy (the second term). Since FQL's Q maximization term only involves the one-step policy π_ω, it is free from BPTT!

As in SAC+BC, the BC coefficient α is the most important hyperparameter in FQL, which needs to be tuned for each task.

Implementation

- Modify `src/scripts/train_offline_online.py` (see TODOs) to combine the offline and online training phases. Note that you will need to initialize a replay buffer and carry over the agent used during offline training into the online training phase. Consider passing a flag for offline/online training into the agent if necessary. Implement `src/agents/fql_agent.py` and `src/agents/sacbc_agent.py`.
- Run your naïve method with both SAC+BC and FQL as your base methods. Run this method on our sanity check environment (`cube-single-play-singletask-task1-v0`). In both cases, you should get above $75-80\%$ success rate in the offline phase, and you must describe the behavior in the online phase.

- SAC+BC (start with α from {30, 100, 300, 1000}) (above 75%)

```
uv run src/scripts/train_offline_online.py --run_group=s1_sacbc --base_config=
    sacbc --env_name=cube-single-play-singletask-task1-v0 --seed=<YOUR SEED> --
    alpha=<YOUR ALPHA> --offline_training_steps 500000 --online_training_steps
    100000
```

- FQL (start with α from {30, 100, 300, 1000}) (above 80%)

```
uv run src/scripts/train_offline_online.py --run_group=s1_fql --base_config=fql
    --env_name=cube-single-play-singletask-task1-v0 --seed=<YOUR SEED> --alpha=<
    YOUR ALPHA> --offline_training_steps 500000 --online_training_steps 100000
```

In future sections, we will just show the options you may want to append to the above commands.

In your report, provide a discussion of these questions

Questions

1. Briefly describe what happens to your loss (and its components) and the Q-values when you transition from the offline to online phases, and why this happens.
2. Was there a difference between how the SAC+BC policy and FQL policy behaved in the offline vs. online finetuning phases? Why?

Deliverables

- ☐ Run your implementation on `cube-single-play-singletask-task1-v0` for both FQL and SAC+BC
- ☐ Plot the training curves for each of the methods, indicating where offline training stops and online training begins.
- ☐ Discussion of questions listed above.

4 Part II: Improving Offline-to-Online RL

Now we will explore several methods to improve beyond the generic methods we implemented in section 3. For this section, you will only be extending FQL (not SAC+BC).

4.1 Using offline data

A very simple trick to improve online finetuning is to include some amount of offline data in the replay buffer. Intuitively, this reduces the distribution shift between the offline and online finetuning phases.

To implement this trick, before starting the online phase, select some data from the offline dataset and insert it into the replay buffer.

4.2 Warm-start RL (WSRL)

Alternatively, we *don't* need to include offline data; we can instead pre-fill the replay buffer with trajectories generated by our offline policy. This is the setup introduced by Warm-start RL (Zhou et al., 2025).

When implementing this trick, we do not update the policy during the first `num_warmup_steps`, saving the data generated in these steps from the policy interacting with the environment, and then after we reach `num_warmup_steps`, we start to update the policy.

You use this trick in the next sections, if you wish.

4.3 IFQL / Rejection Sampling

In HW5, you implemented IQL. We will be implementing the flow version of IQL, originally based on IDQL Hansen-Estruch et al. (2023), and implement rejection sampling.

We use the same objectives as for IQL, with a couple of key differences.

Using a flow policy. Our original objective for IQL is as follows

$$\mathcal{L}(Q) = \sum_{i=1}^{2} \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}} \left[(Q_{\theta_i}(s,a) - r - \gamma V_\psi(s'))^2 \right], \tag{10}$$

$$\mathcal{L}(V) = \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[\ell_2^\tau \left(V_\psi(s) - \min_{i=1,2} \bar{Q}_{\theta_i}(s,a) \right) \right], \tag{11}$$

where $\bar{Q}_{\theta_1}$ and $\bar{Q}_{\theta_2}$ are target networks for Q_{θ_1} and Q_{θ_2}, respectively, $\ell_2^\tau(x) = |\tau - \mathbb{I}(x > 0)|x^2$ is the expectile loss with an expectile parameter $\tau \in [0.5, 1)$ **(which will be the main parameter we tune)**, and $\mathbb{I}(\cdot)$ is the 0-1 indicator function. Originally, AWR (Peng et al., 2019) was used for policy extraction, as follows

$$\mathcal{L}(\pi) = \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[-\min(e^{\alpha A(s,a)}, M) \log \pi_\varphi(a \mid s) \right], \tag{12}$$

$$\text{where } A(s,a) = \min_{i=1,2} Q_{\theta_i}(s,a) - V_\psi(s), \tag{13}$$

We will now modify our actor to be a flow policy, which just changes the objective for the actor (in this version, we do not use a one-step policy, but you can implement this yourself!). Note that we use v_φ to indicate when we are doing partial denoising at a given step t.

$$\mathcal{L}(\pi) = \mathbb{E}_{\substack{(s,a)\sim\mathcal{D}, \\ z\sim\mathcal{N}(0,I_{|\mathcal{A}|}), \\ t\sim\text{Unif}([0,1])}} \left[\frac{1}{|\mathcal{A}|} \left\| v_\varphi(s,\tilde{a},t) - (a - z) \right\|_2^2 \right], \tag{14}$$

$$\text{where } \tilde{a} = (1-t)z + ta. \tag{15}$$

Adding rejection sampling. The other major change for IFQL is using rejection sampling or "best-of-n" sampling. This allows us to evaluate the value of taking different actions generated by our policy and select the best action. We will only use rejection sampling at *test-time*.

Algorithm 1 Best-of-N / Rejection sampling

$N = \#$ of samples

for j in $1 : N$ **do**
  $z^j \sim N(0, I)$
  $a^j_{cand} \sim \pi_\varphi(a|s, z)$
  $v^j_{cand} = \min_{i=1,2} Q_{\theta_i}(s, a^j_{cand})$

end for
$j^* = \arg\max_{j=1,\dots,N} v^j_{\text{cand}}$
$a_{\text{best}} = a^{j^*}_{\text{cand}}$

4.4 Latent Space RL

In the lecture, we discussed latent variable models and how diffusion and flow models can be interpreted as them. While in all past methods we have used RL with a policy that produces actions, in diffusion-space RL (DSRL), we do RL in the latent-noise space. In the original implementation, a fixed BC policy is used, and only a policy that changes the "noise" input to this BC policy is trained. To extend DSRL to the case of an unfixed BC policy, we follow the implementation of DSRL in QAM (Li and Levine, 2026), where we train a BC policy as well.

First, there are two kinds of critics that we learn. The first is our typical Q-function, Q_θ, that is the critic for the BC policy, $\pi^{BC}_\varphi(a|s)$. The second is a critic, Q^z_φ, for our latent-noise policy, π^z_ψ.

$$\mathcal{L}(Q) = \sum_{i=1}^{2} \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}[(y - Q_{\theta_i}(s,a))^2] \quad \text{where} \quad y = r + \frac{\gamma}{2}\sum_{j=1}^{2} \bar{Q}_{\theta_j}(s', a')$$
$$a' \sim \bar{\pi}^{BC}_\varphi(s', \sigma_z z), \quad z \sim \pi^z_\psi(s')$$

$$\mathcal{L}(Q^z) = \sum_{i=1}^{2} \mathbb{E}_{\substack{(s,a,r,s')\sim\mathcal{D} \\ z\sim\mathcal{N}(0,I)}}[(y - Q^z_{\varphi_i}(s,z))^2], \quad \text{where} \quad y = \frac{1}{2}\sum_{j=1}^{2} Q_{\theta_j}(s, a^{BC})$$
$$a^{BC} \sim \bar{\pi}^{BC}_\varphi(s, z), \quad z \sim \mathcal{N}(0, I)$$

$$\mathcal{L}(\pi^{BC}) = \mathbb{E}_{\substack{(s,a)\sim\mathcal{D}, \\ z\sim\mathcal{N}(0,I), \\ t\sim\text{Unif}([0,1])}} \left[\frac{1}{|\mathcal{A}|}\left\|v^{BC}(s, \tilde{a}, t) - (a - z)\right\|_2^2\right], \quad \text{where} \quad \tilde{a} = (1-t)z + ta.$$

$$\mathcal{L}(\pi^z) = \mathbb{E}_{\substack{(s)\sim\mathcal{D} \\ z\sim\pi^z_\psi(s)}} \left[-Q^z_\varphi(s, \sigma_z z)\right] \quad \text{s.t.} \quad \mathbb{E}_{s\sim\mathcal{D}}\left[\mathcal{H}(\pi^z_\psi)\right] \geq \mathcal{H}_{target}$$

$\bar{Q}_{\theta_1}$ and $\bar{Q}_{\theta_2}$ are the target critic networks for our ensemble of critics, Q_{θ_1} and Q_{θ_2}. We also implement a target policy network, $\bar{\pi}^{BC}_\varphi$, for the BC policy, π^{BC}_φ. We update these target networks using the soft update with Polyak averaging, as we did in past homeworks. **We encounter our main hyperparameter, σ_z, which scales the noise samples from π^z_ψ.** $\mathcal{H}_{entropy}$ is usually $-|\mathcal{A}| = -dim(\mathcal{A})$. Let's break these losses down:

- $\mathcal{L}(Q)$: This is our regular TD loss for the action-space critic
- $\mathcal{L}(Q^z)$: This is a distillation loss of our action-space critic into our noise-space critic

- $\mathcal{L}(\pi^{BC})$: This is our regular BC flow loss for our BC flow policy (which is in the action-space)
- $\mathcal{L}(\pi^z)$: We update our policy to maximize its respective Q-function with entropy regularization/constraint

Let's break down that last loss further. We are learning an entropy-regularized SAC policy (see the original SAC paper Haarnoja et al. (2018a)), which is a constrained optimization. However, we can turn this into an unconstrained optimization with a *primal* (update the policy to maximize the Q-function with an entropy constraint term) and *dual* update (update α_η to ensure the constraint is met),

$$\begin{aligned}\textbf{Primal:} \quad \mathcal{L}(\pi^z) &= \mathbb{E}_{z\sim\pi^z_\psi(s)}\left[\alpha_\eta \log \pi^z_\psi(s) - Q^z_\varphi(s, \sigma_z z)\right] \\ \textbf{Dual:} \quad \mathcal{L}(\alpha) &= -\alpha_\eta(\mathbb{E}_{z\sim\pi^z_\psi(s)}\left[\log \pi^z_\psi(s)\right] + \mathcal{H}_{target})\end{aligned}$$

Keep in mind that the primal update has an *expectation over samples*, so the reparameterization trick will need to be applied (i.e. use `rsample()`)!

4.5 Gradients of Q

A common interpretation of diffusion models is that they learn the score function. In the context of a policy $\pi(a|s)$, this score function is

$$\nabla_a \log \pi(a|s)$$

An interpretation of this is that we are learning a policy that iteratively moves in the direction of more "clean" data-like samples. Q-score matching (QSM) (Psenka et al., 2025) proves that by iteratively aligning $\nabla_a \log \pi(a|s)$ to $\nabla_a Q^\pi(s,a)$ we are actually aligning it to $\nabla_a \log \pi^*(a|s)$. Therefore, we can use this as our objective during training to go towards the optimal policy.

We train our model by minimizing the following objectives

$$\begin{aligned}\mathcal{L}(Q) &= \sum_{i=1}^{2} \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}[(y - Q_{\theta_i}(s,a))^2] \\ \text{where} \quad y &= r + \gamma(1-d)\min_{i=1,2} \bar{Q}_{\theta_i}(s', a'), \quad a' \sim \pi_\varphi(s', z), \quad z \sim \mathcal{N}(0, I)\end{aligned}$$

$$\begin{aligned}\mathcal{L}(\pi) &= \mathbb{E}_{\substack{(s,a)\sim\mathcal{D}\\ z\sim\mathcal{N}(0,I)\\ t\sim\text{Unif}([0,T-1])}}\left[||\epsilon_\varphi(s, \tilde{a}_t, t) - \eta\nabla_a Q(s,a)||_2^2\right] + \alpha\mathbb{E}_{\substack{(s,a)\sim\mathcal{D}\\ z\sim\mathcal{N}(0,I)\\ t\sim\text{Unif}([0,T-1])}}\left[||z - \epsilon_\varphi(s, \tilde{a}_t, t)||_2^2\right] \\ \tilde{a}_t &= \sqrt{\hat{\alpha}_t}\cdot a + (\sqrt{1-\hat{\alpha}_t})\cdot z\end{aligned}$$

where ϵ_φ and π_φ are actually the same model. When we use π_φ, this indicates doing the full denoising process to get the denoised actions, while ϵ_φ indicates getting the predicted noise at timestep t in denoising. In the original QSM work, η is a temperature term that scales $\nabla_a Q(s,a)$. In QAM (Li and Levine, 2026), the QSM baseline is implemented with an additional DDPM loss term (second term) with α to balance between the QSM loss and the regular DDPM loss. η **and** α **are the two hyperparameters we will need to tune.** We must also change our sampling process to use DDPM (denoising diffusion probabilistic model) sampling.

We will use the cosine annealing sampling noise schedule (Nichol and Dhariwal, 2021), which determines the

Algorithm 2 DDPM Sampling

$x_T \sim \mathcal{N}(0, I)$,
for t in $T : 0 : -1$ **do**
  $z \sim \mathcal{N}(0, I)$ if $t \neq 1$ else $z = 0$
  $\epsilon_{pred} = \epsilon_\varphi(x_t, s, t)$
  $\mu_\varphi = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\hat{\alpha}_t}} \epsilon_{pred} \right)$
  $\sigma_t = \sqrt{\tilde{\beta}_t}$ Many implementations use β_t instead
  $x_{t-1} = \mu_\varphi + \sigma_t z$

α_t, $\hat{\alpha}_t$ and β_t in the sampling process.

$$\bar{\alpha_t} = \frac{f(t)}{f(0)} f(t) = \cos\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2} \right)^2, \qquad s = 0.08, T = 1$$
$$\beta_t = 1 - \frac{\bar{\alpha_t}}{\bar{\alpha}_{t-1}}$$
$$\alpha_t = 1 - \beta_t$$
$$\hat{\alpha_t} = \prod_{i=1}^{t} \alpha_i$$
$$\hat{\beta}_t = \frac{1 - \hat{\alpha}_{t-1}}{1 - \hat{\alpha}} \beta_t$$

Questions

1. **Adding offline data in online step:** What changes about the data distribution that we are sampling from when we retain offline data?
2. **WSRL:** How is the data distribution different now than when we just retain offline data? Why would this be better?
3. **Rejection sampling:** What is the intuition behind rejection sampling? What are we essentially forcing our action distribution to look like? (*Hint: if we had a random policy and did the same thing, what would the output of rejection sampling look like for this policy?*) What kind of policy do we approximate as $N \to \infty$?
4. **DSRL:** In the original DSRL implementation, we use a fixed BC policy and don't update it, and only update our policy that operates in the latent-noise space. What are the benefits of this RL approach for efficiency, exploration, and access to the model?
5. **QSM:** We can approximate our policy as learning $\pi(a|s) \sim e^{\eta Q(s,a)}$. Why then does our objective $\mathcal{L}(\pi)$ make sense?

Implementation

Use this `BASE COMMAND` and append the commands for each section

```
uv run src/scripts/train_offline_online.py --seed=<YOUR SEED> --offline_training_steps
    500000 --online_training_steps 100000
```

☐ **Adding offline data in online step**

- Add a step before starting your online training that pre-fills the replay buffer with some data from the offline dataset (in `src/scripts/train_offline_online.py`)
- Experiment with various amounts of offline data using FQL as your base method. Run your method on three environments. Your commands might look like this:

- **Sanity check**: This should reach above 80% in the offline phase when you tune $\alpha = \{10, 100, 300, 1000\}$

```
<BASE COMMAND> --group=s2_offline --base_config=fql --env_name=cube-single-
    play-singletask-task1-v0 --offline_data <YOUR AMOUNT OF OFFLINE DATA> --
    alpha <YOUR ALPHA>
```

- `cube-double-play-singletask-task1-v0` - you should be able to reach $> 30\%$ by tuning $\alpha = \{10, 100, 300, 1000\}$ in the offline phase

```
<BASE COMMAND> --group=s2_offline --base_config=fql --env_name=cube-double-
    play-singletask-task1-v0 --offline_data <YOUR AMOUNT OF OFFLINE DATA> --
    alpha <YOUR ALPHA>
```

- `antsoccer-arena-navigate-singletask-task1-v0` - you should be able to reach $> 30\%$ by tuning $\alpha = \{3, 10, 30, 100\}$ in the offline phase

```
<BASE COMMAND> --group=s2_offline --base_config=fql --env_name=antsoccer-
    arena-navigate-singletask-task1-v0 --offline_data <YOUR AMOUNT OF OFFLINE
     DATA> --alpha <YOUR ALPHA>
```

☐ **Warm-start RL**

- Add a condition to only start updating the policy after N warm-up steps (in `src/scripts/train_offline_online.p`
- Experiment with various amounts of data from the offline policy with FQL as your base method.
  - **Sanity check**: This should reach above 80% in the offline phase when you tune $\alpha = \{10, 100, 300, 1000\}$

```
<BASE COMMAND> --group=s2_wsrl --base_config=fql --env_name=cube-single-play-
    singletask-task1-v0 --wsrl_steps <WSRL_STEPS> --alpha <YOUR ALPHA>
```

  - `cube-double-play-singletask-task1-v0` - you should be able to reach $> 30\%$ by tuning $\alpha = \{30, 100, 300, 1000\}$ in the offline phase

```
<BASE COMMAND> --group=s2_wsrl --base_config=fql --env_name=cube-double-play-
    singletask-task1-v0 --wsrl_steps <WSRL_STEPS> --alpha <YOUR ALPHA>
```

  - `antsoccer-arena-navigate-singletask-task1-v0` - you should be able to reach $> 30\%$ by tuning $\alpha = \{3, 10, 30, 100\}$ in the offline phase

```
<BASE COMMAND> --group=s2_wsrl --base_config=fql --env_name=antsoccer-arena-
    navigate-singletask-task1-v0 --wsrl_steps <WSRL_STEPS> --alpha <YOUR
    ALPHA>
```

☐ **IFQL / Rejection sampling**

- Implement IFQL and rejection sampling in `src/agents/ifql_agent.py`. You will also need to update `src/configs/ifql_config.py`.
- Train your method
  - **Sanity check**: You should be able to reach $> 60\%$ by tuning $\tau = \{0.85, 0.9, 0.95\}$ in the offline phase

```
<BASE COMMAND> --group=s2_ifql --base_config=ifql --env_name=cube-single-play
    -singletask-task1-v0 --expectile <YOUR EXPECTILE>
```

  - `cube-double-play-singletask-task1-v0` - you should be able to reach $> 20\%$ by tuning $\tau = \{0.85, 0.9, 0.95\}$ in the offline phase

```
<BASE COMMAND> --group=s2_ifql --base_config=ifql --env_name=cube-double-play
    -singletask-task1-v0 --expectile <YOUR EXPECTILE>
```

- `antsoccer-arena-navigate-singletask-task1-v0` - you should be able to reach $> 15\%$ by tuning $\tau = \{0.85, 0.9, 0.95\}$ in the offline phase

```
<BASE COMMAND> --group=s2_ifql --base_config=ifql --env_name=antsoccer-arena-
    navigate-singletask-task1-v0 --expectile <YOUR EXPECTILE>
```

☐ **DSRL**

- Implement DSRL in `src/agents/dsrl_agent.py` and `src/configs/dsrl_config.py`
- Train your method
  - **Sanity check**: You should be able to reach $> 70\%$ by tuning $\sigma_z = \{0.8, 1.0, 1.2\}$ in the offline phase

```
<BASE COMMAND> --group=s2_dsrl --base_config=dsrl --env_name=cube-single-play
    -singletask-task1-v0 --noise_scale <YOUR NOISE SCALE>
```

  - `cube-double-play-singletask-task1-v0` - you should be able to reach $> 70\%$ by tuning $\sigma_z = \{0.8, 1.0, 1.2\}$ in the offline phase

```
<BASE COMMAND> --group=s2_dsrl --base_config=dsrl --env_name=cube-double-play
    -singletask-task1-v0 --noise-scale <YOUR NOISE SCALE>
```

  - `antsoccer-arena-navigate-singletask-task1-v0` - you should be able to reach $> 5\%$ by tuning $\sigma_z = \{0.8, 1.0, 1.2\}$ in the offline phase

```
<BASE COMMAND> --group=s2_dsrl --base_config=dsrl --env_name=antsoccer-arena-
    navigate-singletask-task1-v0 --noise-scale <YOUR NOISE SCALE>
```

☐ **QSM**

- Implement QSM in `src/agents/qsm_agent.py` and `src/configs/qsm_config.py`.
- Train your method
  - **Sanity check**: You should be able to reach $> 50\%$ by tuning $\eta = \{10, 30, 100\}$ and $\alpha = \{10, 30, 100\}$ in the offline phase

```
<BASE COMMAND> --group=s2_qsm --base_config=qsm --env_name=cube-single-play-
    singletask-task1-v0 --inv_temp <YOUR INV TEMP> --alpha <YOUR ALPHA>
```

  - `cube-double-play-singletask-task1-v0` - You should be able to reach $> 10\%$ by tuning $\eta = \{10, 30, 100\}$ and $\alpha = \{10, 30, 100\}$ in the offline phase

```
<BASE COMMAND> --group=s2_qsm --base_config=qsm --env_name=cube-double-play-
    singletask-task1-v0 --inv_temp <YOUR INV TEMP> --alpha <YOUR ALPHA>
```

  - `antsoccer-arena-navigate-singletask-task1-v0` - you should be able to reach $> 35\%$ by tuning $\eta = \{10, 30, 100\}$ and $\alpha = \{10, 30, 100\}$ in the offline phase

```
<BASE COMMAND> --group=s2_qsm --base_config=qsm --env_name=antsoccer-arena-
    navigate-singletask-task1-v0 --inv_temp <YOUR INV TEMP> --alpha <YOUR
    ALPHA>
```

Deliverables

☐ **Adding offline data in online step**

- Your implementation (submit to autograder on Gradescope)
- Provide two plots that show the success rate over steps for 3 different amounts of offline data on `cube-double-play-singletask-task1-v0` and `antsoccer-arena-navigate-singletask-task1-v0`.
- Discuss the questions provided above.

☐ **Warm-start RL**

- Your implementation
- Pass autograder on Gradescope
- Provide two plots that show the success rate over steps for 3 different amounts of WSRL steps on `cube-double-play-singletask-task1-v0` and `antsoccer-arena-navigate-singletask-task1-v0`.
- Discuss the question provided above.

☐ **IFQL + rejection sampling**

- Your implementation
- Pass autograder on Gradescope
- Discuss the question provided above.

☐ **DSRL**

- Your implementation
- Pass autograder on Gradescope
- Discuss the question provided above.

☐ **QSM**

- Your implementation
- Pass autograder on Gradescope
- Discuss the question provided above.

5 Part III: Implement your own method

Now that we understand how prior work in offline-to-online RL has addressed the challenges that arise when implementing these methods, your task is to improve performance beyond these methods. You may consider:

- Sweeping over hyperparameters (the provided thresholds check your implementation works, but your hyperparameters might not be well-tuned!)
- Re-implementing a method we have not explored in the homework or this project
- Including various tricks from RL works (multiple Qs, action chunking etc.) (see section 9)

Your goals for your method are:

- `cube-single-play-singletask-task1-v0`: This is your sanity check task. Any method you implement will likely be able to saturate this task.
- `cube-double-play-singletask-task1-v0`: Your method should (with hyperparameter tuning) achieve $> 95\%$ averaged over two seeds. This should specifically be your **evaluation success rate after the maximum number of offline and online steps.**
- `antsoccer-arena-navigate-singletask-task1-v0`: Your method should (with hyperparameter tuning) achieve $> 45\%$ averaged over two seeds. This should specifically be your **evaluation success rate after the maximum number of offline and online steps**.

Implementation To add your own agent to your workspace and easily use the same commands as the homework, you can follow these steps:

1. Create a config for your method under `src/configs`
2. Add to the configs in `src/configs/__init__.py` so you can use the command line args
3. Create an agent file in `src/agents`
4. Add to the agents in `src/agents/__init__.py` so you can use the command line args

Deliverables.

- ☐ Implement your method.
- ☐ Pass the autograder on Gradescope
- ☐ Provide two plots of all the methods from the project and compare them to your method on `cube-double-play-singletask-task1-v0` and `antsoccer-arena-navigate-singletask-task1-v0`. Clearly indicate where online training begins in your plot. Choose the best of FQL or SAC+BC from section 3. In total, there should be **7** methods on your plot. Each of your results for your baselines and your method should be reported over **2 seeds**.

5.1 [Optional] Try out your method on harder tasks!

Please only complete this **optional** step after you have completed your project, as you should first put your resources towards the main deliverables. We will open up an Ed thread for you to share your method's performance. We recommend two tasks for you to share your performance on:

- `humanoidmaze-medium-navigate-singletask-task1-v0`: Control a humanoid agent to navigate through a maze.
- `puzzle-4x4-play-singletask-task1-v0`: Solve a puzzle where a robot must push buttons until all the buttons are the same color.

When adding to the Ed thread, please run your method with at least 2 seeds and plot the success rate over steps, clearly indicating the beginning of online training. Share this plot and provide a brief description of your method. We look forward to seeing your submissions!

6 Resources

We provide a reading list (see section 8) and some common tricks (see section 9) as a starting point to explore various existing RL methods that we have not implemented in class. You may also discuss possible ideas during office hours and on Ed.

7 Submission

7.1 Report

Differing from homework submissions, we want this report to be compiled in a research paper style, as if the method you introduce in section 5 is a novel method you are writing about to submit to a conference workshop (**Note: this is very similar to the CS285 custom project option, but we have modified the rubric (see section 7.2.1)**) We will be mostly looking at your method and your experiment section when grading the report.

The report should include the following:

- **Abstract.** This briefly introduces your method and positions it in the field and relative to other methods.
- **Introduction.** This provides more detail on the framing of your method. This can be brief.
- **Related work.** Provide an overview of each of the baselines you've implemented and how your method compares to them.
- **Method.** Provide an overview of the method you implement in section 5 and the key design decisions you make.
- **Experimental Results.** Provide a discussion relevant to each part of this project and then compile your results with your method.
  - **Naïve baselines.** Provide the plots of the naïve offline-to-online implementations from section 3 and the responses to the provided questions.
  - **Improved Baselines.** Provide the plots of the improved offline-to-online baselines from section 4 and the responses to the provided questions.
  - **Comparison of Method to SoTA.** Provide the plots of all the methods in the project to compare to your method. Discuss why your method outperforms these baselines.
- **Discussion.** Summarize the results and describe limitations and future work for your method.

7.2 Rubric

Each section has the following weights for implementation, and finally, a weight for the report. As you will notice, the majority of the weight is in the final report, which will focus on the effort you put into designing your own method. We also allocate a portion of your grade to achieving above the performance threshold.

Component	Points	How
section 3	10%	Complete all deliverables
section 4	$4\% \times 5$ methods $= 20\%$	Complete all deliverables
section 5	20%	Achieve above performance threshold
Method	25%	See extension section in section 7.1
Final Report	25%	See section 7.2.1

7.2.1 Report rubric

While there is a portion of the grade allocated to the performance of your method, we expect you to demonstrate the decisions you made to arrive at your method and a strong understanding of *why* your method works,

which you will describe in the method section of your report. In the rubric, "method" refers to the method you implement in section 5.

	Poor (0-50%)	Fair (50-75%)	Good (75-90%)	Excellent (90-100%)
Extension	Tunes baseline already implemented in project outline with existing hyperparameters.	Introduces and tunes new hyperparameter(s) or capabilities (UTD, number of critics in ensemble, action-chunking etc.) for baseline already implemented in project.	Re-implements and tunes an existing method (not in the project outline or homework).	Re-implements, tunes or expands upon an existing method with a new feature demonstrating strong understanding of concepts applied
Analysis	No analysis of empirical results. No discussion of design decisions for method. Questions provided in outline not answered or are answered poorly or incorrectly.	Some observations about method and comparisons to baselines, but purely empirical, with no justification of design decisions. Questions provided in outline are answered.	Empirical and theoretical backing provided to justify method and results. Questions provided in outline are answered.	In-depth analysis of results and method is well described. Questions provided in outline are answered.
Complete -ness	No results due to incomplete implementation or training failure. Deliverables from section 3 and section 4 are not included.	Implementation with significant issues. Experiments fail to effectively compare method to baselines. Deliverables from section 3 and section 4 are included.	Substantially complete. Experimental results differentiate method but may lack some discussion. Deliverables from section 3 and section 4 are included.	All experiments are complete and results are discussed fully. Deliverables from section 3 and section 4 are included.

7.3 Submitting the Code and Experiment Runs

In order to turn in your code and experiment logs, create a directory that contains the following:

- A directory named `exp` with all the experiment runs from this assignment. **You should choose your best run for each section based on the success rates achieved, and *rename* the directories accordingly as specified below.**
- The `src` folder with all the `.py` files, with the same names and directory structure as the original homework repository.
- There will be separate submissions for each section to ensure it is easy to fit all runs within the size limit (100MB).

There are 5 submission assignments on Gradescope (so you won't easily go over the 100MB limit)

1. **Section 1:** Naive baselines
2. **Section 2.1:** Online retention of offline data + WSRL
3. **Section 2.2:** IFQL + DSRL
4. **Section 2.3:** QSM
5. **Section 3:** Your custom method

You can create these `submit.zip` files using these commands:

```
mkdir s1
zip -r s1/submit.zip exp/s1_sacbc exp/s1_fql src README.md pyproject.toml uv.lock

mkdir s2_1
zip -r s2_1/submit.zip exp/s2_offline exp/s2_wsrl src README.md pyproject.toml uv.lock

mkdir s2_2
zip -r s2_2/submit.zip exp/s2_ifql exp/s2_dsrl src README.md pyproject.toml uv.lock

mkdir s2_3
```

```
zip -r s2_3/submit.zip exp/s2_qsm src README.md pyproject.toml uv.lock

mkdir s3
zip -r s3/submit.zip exp/s3_yours src README.md pyproject.toml uv.lock
```

Follow the structure shown below. For a subdirectory in the `exp` directory (dictated by the tag in your python command), follow this structure `s<X>_<METHOD>` where $X \in \{1, 2, 3\}$ and `METHOD` $\in$ {`sacbc, fql, offline, wsrl, ifql, dsrl, qsm, yours` }. The file structure is provided below:

```
submit.zip
└─ exp
   ├─ s1_sacbc
   │  └─ cube-single
   │     ├─ agent.pt
   │     ├─ eval.csv
   │     ├─ flags.json
   │     ├─ log.pkl
   │     └─ train.csv
   ├─ s1_fql
   │  └─ cube-single
   │     ├─ agent.pt
   │     ├─ eval.csv
   │     ├─ flags.json
   │     ├─ log.pkl
   │     └─ train.csv
   ├─ src
   │  ├─ agents
   │  │  ├─ ...
   │  │  ├─ fql_agent.py
   │  │  ├─ sacbc_agent.py
   │  │  └─ ...
   │  └─ ...
   ├─ pyproject.toml
   ├─ uv.lock
   └─ README.md
```

```
submit.zip
└─ exp
   ├─ s2_offline
   │  ├─ cube-double
   │  │  ├─ agent.pt
   │  │  ├─ eval.csv
   │  │  ├─ flags.json
   │  │  ├─ log.pkl
   │  │  └─ train.csv
   │  └─ antsoccer-arena
   │     ├─ agent.pt
   │     ├─ eval.csv
   │     ├─ flags.json
   │     ├─ log.pkl
   │     └─ train.csv
   ├─ s2_wsrl
   │  ├─ cube-double
   │  │  ├─ agent.pt
   │  │  ├─ eval.csv
   │  │  ├─ flags.json
   │  │  ├─ log.pkl
   │  │  └─ train.csv
   │  └─ antsoccer-arena
   │     ├─ agent.pt
   │     ├─ eval.csv
   │     ├─ flags.json
   │     ├─ log.pkl
   │     └─ train.csv
   ├─ src
   │  ├─ agents
   │  │  ├─ ...
   │  │  ├─ fql_agent.py
   │  │  ├─ ifql_agent.py
   │  │  ├─ dsrl_agent.py
   │  │  ├─ qsm_agent.py
   │  │  └─ ...
   │  └─ ...
   ├─ pyproject.toml
   ├─ uv.lock
   └─ README.md
```

```
submit.zip
└─ exp
   ├─ s2_ifql
   │  ├─ cube-double
   │  │  ├─ agent.pt
   │  │  ├─ eval.csv
   │  │  ├─ flags.json
   │  │  ├─ log.pkl
   │  │  └─ train.csv
   │  └─ antsoccer-arena
   │     ├─ agent.pt
   │     ├─ eval.csv
   │     ├─ flags.json
   │     ├─ log.pkl
   │     └─ train.csv
   ├─ s2_dsrl
   │  ├─ cube-double
   │  │  ├─ agent.pt
   │  │  ├─ eval.csv
   │  │  ├─ flags.json
   │  │  ├─ log.pkl
   │  │  └─ train.csv
   │  └─ antsoccer-arena
   │     ├─ agent.pt
   │     ├─ eval.csv
   │     ├─ flags.json
   │     ├─ log.pkl
   │     └─ train.csv
   ├─ src
   │  ├─ agents
   │  │  ├─ ...
   │  │  ├─ fql_agent.py
   │  │  ├─ ifql_agent.py
   │  │  ├─ dsrl_agent.py
   │  │  ├─ qsm_agent.py
   │  │  └─ ...
   │  └─ ...
   ├─ pyproject.toml
   ├─ uv.lock
   └─ README.md
```

```
submit.zip
└─ exp
   ├─ s2_qsm
   │  ├─ cube-double
   │  │  ├─ agent.pt
   │  │  ├─ eval.csv
   │  │  ├─ flags.json
   │  │  ├─ log.pkl
   │  │  └─ train.csv
   │  └─ antsoccer-arena
   │     ├─ agent.pt
   │     ├─ eval.csv
   │     ├─ flags.json
   │     ├─ log.pkl
   │     └─ train.csv
   ├─ src
   │  ├─ agents
   │  │  ├─ ...
   │  │  ├─ fql_agent.py
   │  │  ├─ ifql_agent.py
   │  │  ├─ dsrl_agent.py
   │  │  ├─ qsm_agent.py
   │  │  └─ ...
   │  └─ ...
   ├─ pyproject.toml
   ├─ uv.lock
   └─ README.md
```

```
submit.zip
└─ exp
   ├─ s3_yours
   │  ├─ cube-double
   │  │  ├─ agent.pt
   │  │  ├─ eval.csv
   │  │  ├─ flags.json
   │  │  ├─ log.pkl
   │  │  └─ train.csv
   │  └─ antsoccer-arena
   │     ├─ agent.pt
   │     ├─ eval.csv
   │     ├─ flags.json
   │     ├─ log.pkl
   │     └─ train.csv
   ├─ src
   │  ├─ agents
   │  │  ├─ ...
   │  │  ├─ your_agent.py
   │  │  └─ ...
   │  └─ ...
   ├─ pyproject.toml
   ├─ uv.lock
   └─ README.md
```

Turn in your assignment on Gradescope. Upload the zip file with your code and log files to **Default Final Project Section 1, 2.1, 2.2, 2.3, 3** and upload the PDF of your report to **Default Final Report**.

8 Reading list

There are various works you might want to look into to get started. Here are some of them:

- RLPD Ball et al. (2023)
- CQL Kumar et al. (2020)
- Q-chunking Li et al. (2025)
- QAM Li and Levine (2026)
- AWAC Nair et al. (2021)
- Balancing offline and online data Lee et al. (2021)

9 Tricks to consider

Here are some tricks you might consider implementing regardless of your method

- **Using pessimistic Q-values.** Instead of taking the mean or the min, we can take $Q = \frac{1}{N}\sum_{i=1}^{N} Q_i(s,a) - \rho \cdot \text{VAR}(Q_i)$ (
- **Increasing number of critics.** Instead of just 2 critics in the ensemble, you can experiment with more!
- **Rejection sampling.** Try implementing rejection sampling beyond IFQL
- **Increasing the update-to-data (UTD) ratio.** Instead of doing one update for each new sample from the environment, you can increase the number of updates to get more juice out of a given sample
- **Action chunking.** Instead of using actions over one step, you can extend to using an action chunk that predicts actions over a given horizon (see Li et al. (2025))

References

Philip J. Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data, 2023. URL https://arxiv.org/abs/2302.02948.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, G. Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *ArXiv*, abs/1812.05905, 2018b.

Philippe Hansen-Estruch, Ilya Kostrikov, Michael Janner, Jakub Grudzien Kuba, and Sergey Levine. Idql: Implicit q-learning as an actor-critic method with diffusion policies, 2023. URL https://arxiv.org/abs/2304.10573.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning, 2020. URL https://arxiv.org/abs/2006.04779.

Seunghyun Lee, Younggyo Seo, Kimin Lee, Pieter Abbeel, and Jinwoo Shin. Offline-to-online reinforcement learning via balanced replay and pessimistic q-ensemble, 2021. URL https://arxiv.org/abs/2107.00591.

Qiyang Li and Sergey Levine. Q-learning with adjoint matching, 2026. URL https://arxiv.org/abs/2601.14234.

Qiyang Li, Zhiyuan Zhou, and Sergey Levine. Reinforcement learning with action chunking, 2025. URL https://arxiv.org/abs/2507.07969.

Clare Lyle, Zeyu Zheng, Khimya Khetarpal, James Martens, Hado van Hasselt, Razvan Pascanu, and Will Dabney. Normalization and effective learning rates in reinforcement learning, 2024. URL https://arxiv.org/abs/2407.01800.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets, 2021. URL https://arxiv.org/abs/2006.09359.

Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models, 2021. URL https://arxiv.org/abs/2102.09672.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. In *International Conference on Learning Representations (ICLR)*, 2025.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning, 2019. URL https://arxiv.org/abs/1910.00177.

Michael Psenka, Alejandro Escontrela, Pieter Abbeel, and Yi Ma. Learning a diffusion model policy from rewards via q-score matching, 2025. URL https://arxiv.org/abs/2312.11752.

Zhiyuan Zhou, Andy Peng, Qiyang Li, Sergey Levine, and Aviral Kumar. Efficient online reinforcement learning fine-tuning need not retain offline data, 2025. URL https://arxiv.org/abs/2412.07762.