# Test-Time Gradient Guidance of Flow Policies in Reinforcement Learning

**Zhiyuan Zhou**[1], **Andy Peng**[1], **Charles Xu**[1], **Qiyang Li**[1], **Tobias Springenberg**[2], **Kevin Frans**[1], **Sergey Levine**[1,2]
[1] UC Berkeley, [2] Physical Intelligence

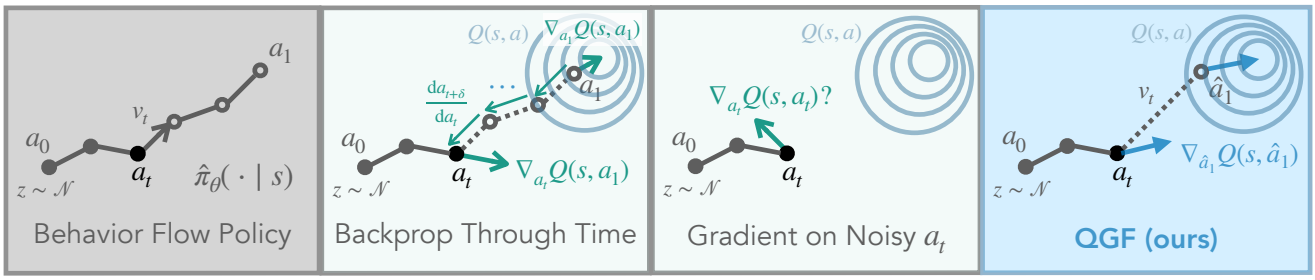


**Figure 1:** We propose QGF (Q-Guided Flow), an RL algorithm that guides denoising steps of a policy trained via flow matching with critic gradient at test time. Our critic gradient estimator avoids both taking gradient at noisy action not seen during training and performing expensive, high-variance backpropagation through time, and performs better than other test-time RL methods while being competitive with the best train-time baseline.

**Expressive continuous control policies, such as diffusion and flow models, form the backbone of recent advances in scaling imitation learning for simulated and real robot control. While they are known to scale stably in the supervised imitation learning setting, incorporating them into reinforcement learning (RL) pipelines for policy improvement has proven more difficult. It often requires specialized training objectives or backpropagating through denoising processes, which cause well-known issues with stability and affect scalability. In this paper we study the question of whether simple policy improvement schemes at test time alone, leaving stable supervised policy training intact, can be a competitive alternative which sidesteps these issues. To this end, we propose QGF (Q-Guided Flow), an RL algorithm that performs policy optimization entirely at test time. QGF works by pre-training both a reference flow policy (via a standard behavioral cloning objective) and a value function critic and, at test time, using the value gradient to guide the reference policy to generate higher-value actions without any additional policy learning. Empirically, QGF outperforms prior test-time RL methods on single-task and goal-conditioned offline RL benchmarks with high-dimensional action spaces, and is competitive with state-of-the-art training-time algorithms while being much cheaper to run. Moreover, it exhibits favorable scaling with model size by avoiding the instability of actor-critic training, offering a practical and effective alternative RL algorithm with expressive policies. Code:** github.com/zhouzypaul/qgf

## 1. Introduction

Reinforcement learning (RL) is a powerful framework for optimizing reward-maximizing behavior, but scaling RL algorithms—especially in the offline or off-policy setting—remains a significant challenge. A key difficulty lies in the complexity and instability of policy optimization: unlike simple supervised learning objectives that optimize towards a known fixed target, RL typically alternates between learning a value function and optimizing an actor to maximize this learned function. This coupled training dynamic makes RL policy training sensitive to hyperparameters and prone to optimization instability [61, 16, 21], making it hard to scale. This challenge is particularly acute when using expressive policy classes such as diffusion and flow models. These models are attractive because they can represent complex, multimodal action distributions and have been shown to scale effectively when performing standard supervised learning via behavioral cloning (BC). However, incorporating

them into RL pipelines typically requires designing specialized training objectives [5, 50, 28, 4] or backpropagating through long denoising processes [63, 20, 9, 69, 62, 11, 40], which undermines their scalability and stability.

This tension motivates an alternative approach: *rather than designing a more scalable RL objective for policy optimization, can we train the actor with standard supervised learning and use test-time compute to optimize it against a value function?* Concretely, we want to train a policy with standard behavior cloning (we will refer to this policy as the reference policy in the following), and learn a critic separately using standard temporal-difference (TD) learning methods. Then, at inference time, we want to use the critic to "guide" the reference policy towards higher-value actions, without changing the policy parameters. Such an approach would then avoid the instability of needing to train an actor to maximize a changing critic, but instead directly leverage well-understood and scalable supervised objectives during policy training, and apply reward optimization dynamically at test time.

A straightforward way to implement this idea is to combine sampling from the reference policy with optimization of actions against the learned critic. Previous successful attempts to use test-time compute in this way have mostly relied on a best-of-N (BFN) sampling strategy [57, 19, 35], where one draws $N$ actions from the policy and then picks the value-maximizing action according to the critic. However, BFN sampling can be prohibitively expensive in high-dimensional action spaces. One might wonder whether, instead of sampling, we can rely on the critic gradient $\nabla_a Q(s, a)$ to directly steer action generation towards high-value directions. Unfortunately, gradient-guidance methods are not straightforward to implement for flow and diffusion models, since actions are generated by an iterative denoising process. The most naïve way to guide this denoising process is to use the gradient of $Q$ with respect to intermediate "noisy" actions by backpropagating through the entire denoising process [63, 20, 9, 69, 62, 11, 40], which is computationally expensive, unstable [50], and has high variance (see Fig. 3). One cannot simply avoid such backpropagation because the critic has not been trained to handle intermediate noisy actions, and directly using gradients there leads to biased guidance (see Fig. 2).

In this paper, we introduce a method, QGF (Q-Guided Flow), that effectively utilizes critic guidance during sampling from flow policies. QGF circumvents backpropagation through time and critic gradient at untrained noisy actions by using the critic gradient at an approximate action derived from following the learned velocity flow field for a single, large, Euler integration step (see Fig. 1). We conduct extensive empirical analysis in an offline RL setting (both single-task and goal-conditioned) and show that QGF achieves strong performance compared to both other test-time RL algorithms and state-of-the-art RL algorithms that explicitly train policies to achieve high reward values. In addition, we find that QGF scales well as we increase the model size, since we avoid optimizing the policy against an evolving critic during training, and only train with a stable supervised learning loss. Overall, we establish that gradient-based test-time RL is a promising direction for optimizing generative action models that is stable, scalable, and easy to use in practice.

## 2. Related Work

**Offline RL** studies how to learn a reward-maximizing policy solely from a fixed dataset without any interactions and environment feedback. The key challenge is to extract optimal behavior from a mixed-quality offline dataset, while preventing erroneous extrapolation that deviates too far from dataset behavior. Prior work has extensively studied both value learning methods that prevent erroneous extrapolations [2, 17, 29, 31, 66] and policy extraction techniques [3, 15, 19, 44, 51, 59, 64]. The predominant approach in this setting optimizes the policy to maximize rewards during training, and directly samples from such policy at test time. Our approach, on the other hand, does not train the policy to be reward-seeking. In contrast, we train the policy with standard behavioral cloning to imitate the policy that collected the mixed-quality dataset, and perform policy optimization entirely at test time, with guidance from the value function gradient.

**Flow and diffusion policies for RL.** Previous works have explored different ways of using expressive models such as diffusion and flow models to train the policy for continuous control in RL, including policy gradient methods [54, 43], importance-weighted methods [52, 12, 8, 70, 4, 67], and actor-critic methods [63, 20, 9, 69, 62, 40, 5, 50, 28, 4]. Using these iterative denoising methods for actor-critic style algorithms is especially challenging since the actor $\pi_\theta(a \mid s)$ is trained with the critic gradient $\nabla_\theta Q(s, \pi_\theta(a \mid s))$. For single-step Gaussian policies, the actor can leverage the gradient through the reparameterization trick [18]; for flow and diffusion policies however,

leveraging the gradient requires backpropagating through a multi-step denoising process [63, 20, 9, 69, 62, 40], which is expensive and can be unstable in practice [50]. Previous works have attempted to avoid unstable backpropagation by (1) distilling the multi-step policy into a single-step policy [50, 4], (2) using a single-step Euler-integration approximation of the multi-step denoising process [25], or (3) simply ignoring the denoising process and taking the critic gradient at noisy actions $a_t$ (*i.e.*, $\nabla_{a_t} Q(s, a_t)$) [53, 68, 13, 36]. Approach (1) requires training a separate distilled network, which is expensive and can suffer from expressivity issues; approach (2) is much cheaper and doesn't require additional training, but can suffer from approximation error, though we find it to perform better in practice (see Fig. 4); approach (3) is the simplest, but leads to biased guidance (see Fig. 2). Most similar to our work is EDP [25], which takes approach (2) above. We also use a single-step Euler integration to approximate the denoising process. However, we leverage the critic gradient with respect to the action, while EDP uses the critic gradient with respect to policy parameters. More importantly, EDP requires training the actor with RL objectives at training time, whereas our method pre-trains a value function and uses its gradient to optimize policy actions at test-time during the action denoising process of a behavior flow policy.

**Test-time methods in RL** often focus on using a value function to refine actions from a reference policy at test time. This can be done by either using the critic to rank action samples and pick the best one [35, 19], or hill-climbing on the critic value with additional gradient steps on fully denoised actions [41, 65]. Another line of methods can also control the optimality of the policy at test time not by using the critic, but by training a policy that is conditioned on optimality [14, 30]. Most similar to our work is QFQL [24], which uses the critic gradient to guide the action denoising process. However, QFQL takes the critic gradient at a noisy action step (*i.e.*, $\nabla_{a_t} Q(s, a_t)$), where the noisy action may often go out-of-distribution for the critic that has only been trained on fully denoised actions. In contrast, our method leverages the critic gradient directly on approximations of fully denoised actions, and performs better in practice (see Figs. 5 and 8).

## 3. Preliminaries and Problem Setting

**Markov decision process and Q-learning.** We consider a Markov Decision Process (MDP) [58] $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(s'|s, a) : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function, $r(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\gamma \in [0, 1)$ is the discount factor. The goal is to learn a policy $\pi_\theta : \mathcal{S} \to \Delta(\mathcal{A})$ that maximizes expected discounted return: $J(\pi) = \mathbb{E}_{a_t \sim \pi(\cdot|s_t), s_{t+1} \sim P(\cdot|s_t, a_t)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$. The $Q$-function approximates the expected discounted return from state $s$ after taking action $a$ and following the policy $\pi$: $Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a\right]$. $Q$-functions are typically trained by minimizing the temporal difference (TD) loss: $\mathbb{E}_{s,a,s' \in \mathcal{D}}(Q(s, a) - r(s, a) - \gamma \mathbb{E}_{\hat{a}_1 \sim \pi}[Q_{\bar{\phi}}(s', \hat{a}_1)])^2$. Notably, this loss relies on sampling from $\pi$. Since we are interested in finding $\pi$ only at test time, we instead rely on Implicit Q-learning (IQL) [29], which allows us to learn a Q-function for near optimal policies using actions from the dataset alone. In particular, IQL minimizes loss $\mathcal{L}_Q(\phi) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[(r(s, a) + \gamma V_\psi(s') - Q(s, a))^2\right]$, where $V_\psi$ is a state value function trained to regress to the upper expectile of $Q$: $\mathcal{L}_V(\psi) = \mathbb{E}_{(s,a) \sim \mathcal{D}}[L_2^\tau \left(Q(s, a) - V_\psi(s)\right)]$. Here $L_2^\tau$ is the expectile regression loss $L_2^\tau(u) = |\tau - \mathbf{1}(u < 0)| \, u^2$ and $\tau \in (0.5, 1]$.

**Behavior-regularized RL.** In the *offline RL* setting, the agent does not have access to the environment and must instead learn from a fixed dataset $\mathcal{D} = \{(s_i, a_t, s_i', r_i)\}_{i=1}^{|D|}$ of transitions collected by a behavior policy $\hat{\pi}$. Policy optimization is then often performed by both maximizing $Q$-values while regularizing the policy towards the data distribution. Regularization is necessary since, without access to the environment, the policy can exploit out-of-distribution $Q$-values that are never corrected by real experience. To address this, methods regularize the policy from deviating too much from the behavior policy's distribution. Specifically, we consider the KL-regularized RL objective with respect to the behavior policy $\hat{\pi}$ where $\beta > 0$ controls the strength of the behavioral regularization:

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right] - \beta \, \mathbb{E}_{s \sim d^{\pi_\theta}(s)} \left[D_{\mathrm{KL}}(\pi_\theta(\cdot \mid s) \,\|\, \hat{\pi}(\cdot \mid s))\right]. \tag{1}$$

**Flow matching.** Flow matching [39] generative models are parameterized by a time-dependent velocity field $v_\theta(x,t) : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$ that transports samples from a simple noise distribution $p_0 = \mathcal{N}(0, I_d)$ to a target data distribution $p_1 = p(x)$ via an ODE: $\mathrm{d}\hat{x}_t = v_\theta(\hat{x}_t, t)\,\mathrm{d}t, \quad \hat{x}_0 \sim p_0$. The velocity field is trained to match the direction of linear interpolants $x_t = (1-t)x_0 + tx_1$ between noise and data samples via flow matching:

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{t\sim\mathcal{U}[0,1],\, x_0\sim p_0,\, x_1\sim p_1}\left[\|v_\theta(x_t,t) - (x_1 - x_0)\|_2^2\right]. \tag{2}$$

To use flow matching for policy training, we condition the velocity field on the state $s$ and train on dataset actions as targets. At inference time, actions are generated by integrating the ODE from $\hat{x}_0 \sim p_0$ to $\hat{x}_1$, which serves as a valid sample from the policy distribution $\pi_\theta(a \mid s)$. Throughout the paper, we will refer to samples from the policy as "denoised actions", and the intermediates produced by integration as "noisy actions".

## 4. Test-Time Gradient Guidance in RL

Recall that our goal is to optimize the KL-regularized reward maximization objective in Eq. (1). It is well-known in the literature [6, 1, 51] that the solution to this optimization problem is given via the closed form:

$$\pi(a \mid s) \propto \hat{\pi}(a \mid s) \cdot \exp(Q(s,a))^{1/\beta}, \tag{3}$$

where we take the reference policy $\hat{\pi}(a \mid s)$ to be a flow policy that is trained via behavioral cloning loss in Eq. (2).

We would like to transform this equation into one that tells us how to sample from high likelihood regions of $\pi$ by making use of the gradient $\nabla_a Q(s,a)$. Unfortunately, the likelihood of our reference policy $\hat{\pi}(a|s)$ (represented by a flow matching or diffusion model) cannot be evaluated exactly. However, diffusion (and under some constraints flow matching) can be interpreted as optimizing a lower bound on the log-likelihood [27]; and diffusion models learn a score function directly. Using this perspective (assuming the flow corresponds to the score function $\nabla_a \log \hat{\pi}(a|s)$) we can write the score for $\pi$ from Eq. (3) as

$$\nabla_a \log \pi(a \mid s) = \nabla_a \log \hat{\pi}(a \mid s) + 1/\beta \cdot \nabla_a Q(s,a). \tag{4}$$

Additionally, following prior work [13], we can extend this definition to an extended action space with noisy perturbations of actions stemming from the intermediate steps of integrating the diffusion denoising process:

$$\nabla_{a_t} \log \pi(a_t \mid s) \approx \nabla_{a_t} \log \hat{\pi}(a_t \mid s) + 1/\beta \cdot \nabla_{a_t} Q(s,a_t), \tag{5}$$

where $a_t$ corresponds to the partially denoised action at denoising step $t$. This suggests that in order to sample from the improved policy $\pi$, we can modify the denoising process to integrate over the score function of the reference policy plus a guidance term $\nabla_{a_t} Q(s,a_t)$. A higher weight on the guidance term leads to a policy that is less constrained by the reference policy, and maximizes the $Q$ function to a higher degree. This is analogous to performing classifier guidance [7] with a learned Q function replacing the classifier.

**Problems with naïve Q gradient guidance.** The guidance term in Eq. (5) is the gradient of the $Q$-function at noisy actions $a_t$. This gradient can be unreliable since the critic $Q(s,a)$ is only trained on the denoised action space and querying it at out-of-distribution noisy actions may require the gradient of the Q-function to be correct far from its training data, which is not generally guaranteed. We therefore refer to $\nabla_{a_t} Q(s,a_t)$ as the "OOD gradient". Taking the gradient at noisy action $a_t$ is also problematic because $a_t \in \mathbb{R}^{|\mathcal{A}|}$ might not even correspond to a valid action in $\mathcal{A}$.

Alternatively, we could consider ways to query Q on denoised actions only. Since flow matching deterministically maps each noisy action $a_t$ to a denoised action $a_1$ by integrating through the ODE (*i.e.*, $a_1 = \mathrm{ODE}(a_t) = a_t + \int_t^1 v_\theta(a_\tau, \tau) d\tau$), we can interpret the Q value of a noisy action to be the Q value of its denoised version:

$$Q(s,a_t) := Q(s, \mathrm{ODE}\,(a_t)). \tag{6}$$

Thus, a more principled way would be to use $\nabla_{a_t} Q(s, \mathrm{ODE}\,(a_t))$. Computing this gradient requires back propagation through the denoising process, which we call the "BPTT gradient". While more principled, computing

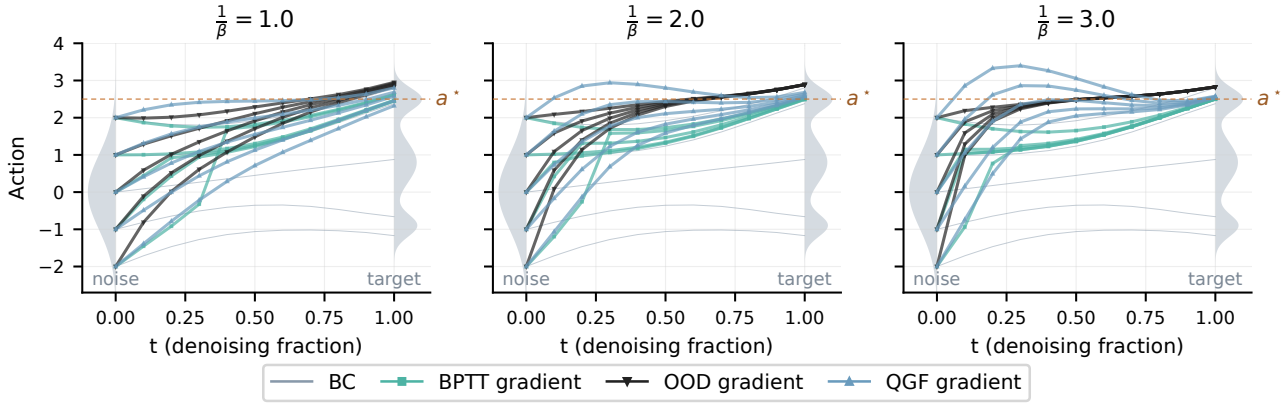


**Figure 2:** Illustrative example of 1D denoising process mapping Gaussian noise to a tri-modal distribution, with $Q$ defined as negative L2 distance to the optimal action $a^*$. We compare the base BC flow and three critic-gradient guidance methods (BPTT, OOD, QGF) across three guidance weights. While BPTT and QGF converge to $a^*$, **guidance with the OOD gradient $\nabla_{a_t} Q(s, a_t)$ does not result in the optimal solution**. Further, Fig. 14 shows **BPTT can be highly unstable**.

this gradient is expensive. Furthermore, we find that the BPTT gradient is very sensitive to noise and has a high variance. This effect is clearly visible in Fig. 3, where we plot the cosine similarity between the gradient evaluated at $a_t$ and a slightly perturbed noisy action $a_t + \epsilon$, *i.e.*, $\cos(G(s, a_t), G(s, a_t + \epsilon))$ where $G$ corresponds to the estimated gradient. Values closer to 1 correspond to gradient estimators that are less sensitive to noise.

**Illustrative example of suboptimal $Q$-gradient guidance.** We show a simple illustrative example in which using either of the above described gradient estimators results in suboptimal denoised actions. In Fig. 2, we visualize a 1D denoising process where a flow matching model tries to map input noise distribution to a tri-modal action distribution. The "Q-function" in this didactic example is the negative squared distance to the optimal action. We visualize the analytical solution to the base behavioral cloning (BC) flow and the solution when it is guided by three different gradients: the OOD gradient $\nabla_{a_t} Q(s, a_t)$, the BPTT gradient $\nabla_{a_t} Q(s, \text{ODE}(a_t))$, and the approximate gradient we propose in Section 5. As we can see, regardless of the guidance weight, using the OOD action gradient always misguides the flow and "bias" it to a suboptimal action after denoising. Section 6 also corroborates this illustrative example and shows that test-time guidance with the OOD gradient (following [24]) results in poor performance. While the BPTT gradient is less "biased", it is still highly sensitive to noise (see Fig. 3) and therefore leads to instability for policy optimization, as we find in Fig. 14. On the other hand, the gradient estimator that we propose below, QGF, has much smaller variance (see Fig. 3) and results in stable optimization towards the optimal action.

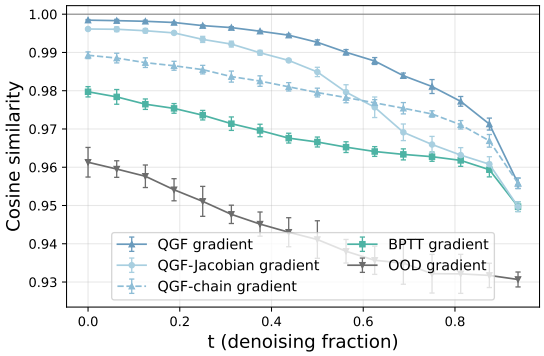


**Figure 3:** Sensitivity of different gradient estimators to noise in the action space: for each gradient estimator $G$, the plot shows the cosine similarity between $G(s, a_t)$ and $G(s, a_t + \epsilon)$. Our proposed gradient estimator has the least variance and least sensitivity to noise. Averaged over 20 tasks and 4 seeds.

## 5. Q-Guided Flow

In this section, we will show that we can derive a gradient estimator that: i) does not query the critic on OOD inputs, ii) is cheaper than BPTT through ODE integration, and iii) has lower variance and consequently more effectively optimizes actions to maximize $Q$-values. We start by noting that, instead of fully integrating the denoising ODE, we can obtain a cheap, first-order approximation to the ODE solution by taking a single large

Euler integration step, following the local velocity field at action $a_t$ all the way to a denoised action:

$$\hat{a}_1 = a_t + v_\theta(s, a_t, t) \cdot (1 - t), \tag{7}$$

where $v_\theta(s, a_t, t)$ is the flow velocity function of the reference policy $\hat{\pi}(a \mid s)$ at time step $t$. As we will show in Fig. 4, using this first-order approximation is not just a convenience. Perhaps surprisingly, it is actually *more* effective than running the full denoising process because it is less constrained to the exact dataset distribution. From this, we can approximate the ground truth gradient as

$$\nabla_{a_t} Q(s, a_1) \approx \nabla_{a_t} Q(s, \hat{a}_1) = \left( \frac{\partial \hat{a}_1}{\partial a_t} \right)^\top \nabla_{\hat{a}_1} Q(s, \hat{a}_1), \tag{8}$$

which is a product between the gradient of $Q$ and the Jacobian of the denoised action with respect to the noisy action, $J = \frac{\partial \hat{a}_1}{\partial a_t}$. Empirically, we find that $J$ can be ill-behaved since it requires differentiating through $v_\theta(s, a_t, t)$, and replacing it with the identity entirely ($J \approx \hat{J} := I$) yields better performance, effectively computing

$$\nabla_{a_t} Q(s, a_1) \approx \hat{J}^\top \nabla_{\hat{a}_1} Q(s, \hat{a}_1) \quad \text{where} \quad \hat{a}_1 = a_t + v_\theta(s, a_t, t) \cdot (1 - t),\ \hat{J} = I. \tag{9}$$

In summary, we use a first-order approximation to denoise action $a_t$, and take the critic gradient at the denoised action. QGF uses this $Q$ gradient to guide the denoising of a reference policy during inference time, essentially adding a weighted value of this gradient to the velocity of the reference policy to steer it towards more optimal actions. The full algorithm is described in Algorithm 1. Two design choices in QGF may appear to be crude approximations: we drop the Jacobian entirely and use a first-order approximation of the denoised action. Both look like approximations one would only tolerate for efficiency. Surprisingly, we find that neither is merely a compromise and both outperform their more "exact" counterparts. In the analysis below, we show that these choices yield a lower-variance gradient estimator that is better at optimizing $Q$-values.

**Algorithm 1:** QGF (Q-Guided Flow) inference.

**Inference time**

**Input:** state $s$, ref. flow $v_\theta$, $Q$, guid. weight $\frac{1}{\beta}$, $\delta = 1/T$
$a_0 \sim \mathcal{N}(0, I)$
**for** $t = 0, \delta, 2\delta \ldots, 1 - \delta$ **do** ◂ *flow denoising*
  $\hat{a}_1 \leftarrow a_t + (1 - t)\, v_\theta(s, a_t, t)$
  $g \leftarrow \nabla_{\hat{a}_1} Q(s, \hat{a}_1)$ ◂ *QGF grad. est.*
  $a_{t+\delta} \leftarrow a_t + \delta \cdot \left( v_\theta(s, a_t, t) + \frac{1}{\beta}\, g \right)$ ◂ *Q guidance*
**end for**
**return** $a_1$

**Analysis of the Jacobian.** We shall refer to Eq. (9) as the QGF estimator and Eq. (8) as the QGF-Jacobian estimator. Calculating the Jacobian essentially requires taking partial derivative through $v_\theta$, and can be ill-behaved at times, e.g., in early steps when Eq. (7) is a crude approximation of the ground truth denoising process. We empirically find two ways to address this: regularizing the Jacobian (see Fig. 15) or replacing it with the identity. Indeed, Fig. 3 shows that including the Jacobian in the gradient estimator can make it more sensitive to noise in the input. This high sensitivity to noise in turn makes QGF-Jacobian a worse "optimizer" for $Q$-values than QGF: we can view each gradient estimator that guides flow denoising as an "optimizer" that tries to optimize for high $Q$ value actions, which we find to roughly correlate with performance. Fig. 4 shows the $Q$ value of the final denoised actions when following different gradient guidance during denoising, and shows QGF achieves a higher $Q$ value than QGF-Jacobian. We compare both $J = \frac{\partial \hat{a}_1}{\partial a_t}$ and $J \approx I$ in our experiments below, and find that in fact not using the Jacobian performs better (see Fig. 5), especially on hard environments (see Fig. 8). We note that there is a large body of work that shows that such approximate gradients are often sufficient [37, 46, 23, 32, 42].

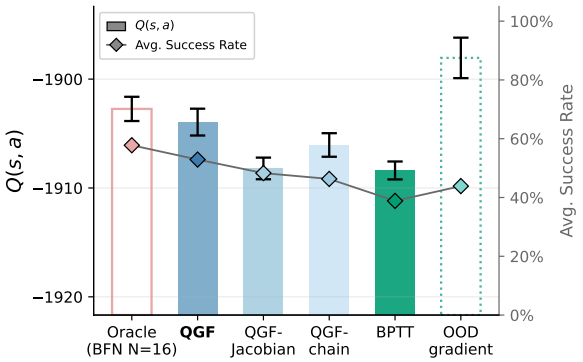


**Figure 4:** $Q$-values of the denoised action with different gradient guidance, which roughly correlate with performance. Averaged over 20 OGBench environments. **QGF is the best gradient-based optimizer, getting close to the performance of the best-of-n oracle.**

**First-order approximation.** To analyze the impact of the first-order approximation, we compare to a variant where the denoised action is obtained by following the entire denoising chain: $a_1 = \text{ODE}(a_t)$. We compare both gradient estimators ($\nabla_a Q(s,a)|_{a=\hat{a}_1}$ v.s. $\nabla_a Q(s,a)|_{a=a_1}$) without the Jacobian (*i.e.*, $J \approx I$) since that leads to better performance and avoids the instability of BPTT. We will refer to this variant as the QGF-chain estimator. Similarly, we find that QGF-chain is more sensitive to noise in Fig. 3, making it also a worse optimizer for $Q$-values in Fig. 4. Indeed, we find in Fig. 15 that QGF-chain underperforms QGF. We hypothesize this is due to QGF being better at mode selection: following the full denoising process of the base BC flow restricts the denoised action to cover the full dataset distribution, while QGF allows deviation from the exact dataset distribution and allows the flow to choose only certain modes of the dataset distribution. This ability to choose good modes enables QGF to guide a base BC policy to select better actions at test time.

In addition, Fig. 4 also shows QGF to be a better Q-optimizer than all the other gradient-based estimators except for the OOD gradient, which we find to exploit the $Q$ function with OOD actions and does not lead to good performance (see Section D for a detailed discussion). Overall, our analyses shed some light on why QGF is a particularly effective gradient-estimator for test-time guidance.

**Implementation details.** QGF is a test-time RL algorithm, using an approximated critic gradient in Eq. (9) to guide action denoising. The algorithm is agnostic to the training of the reference policy and the critic. For simplicity, in this paper, we train the reference policy with behavioral cloning and the value function with IQL [29]. We purposely chose an in-sample value-learning algorithm for critic training so that we can fully decouple value learning and policy extraction. See Section 6.5 for using QGF with other types of value functions.

## 6. Experimental Results

We conduct experiments to study how well QGF is able to optimize the policy at test-time on a range of long-horizon challenging manipulation tasks. We focus on manipulation domains because offline datasets in these environments typically have lower state-action coverage and are therefore harder for RL algorithms. Concretely, we study the following research questions: **(1)** Can QGF optimize the policy well compared to other RL methods? **(2)** Can more test-time compute increase the performance for QGF? **(3)** Can QGF be scaled to solve hard tasks in goal-conditioned RL and utilize larger model architectures? **(4)** Can QGF work with different types of critics?

### 6.1. Experimental Setup

We consider seven different environments (5 tasks each) from OGBench [47] in the offline RL setting: `scene`, `puzzle-4x4` (`p44`), `puzzle-4x5` (`p45`), `puzzle-4x6` (`p46`), `cube-triple` (`c3`), `cube-quadruple` (`c4`), `cube-octuple` (`c8`). We follow the action chunking setting [35] with a chunk size of $h = 5$. In this setting, the policy outputs a sequence of $h$ actions and executes them one by one in the environment. This high-dimensional action space induces a much more complex distribution compared to using single-step atomic actions, and therefore makes an ideal testbed to compare policy extraction methods. Since learning from the default OGBench datasets (ranging from 1M to 3M transitions in size) has saturated performance for offline RL with action chunking [33], we use the datasets with 100M transitions [35]. These larger datasets allow for a bigger jump in performance (compared to the data collection policy). To stress test our method we also consider the 1B transitions dataset for `p46` and `c8`. Finally, we consider a challenging sparse reward setting for the harder puzzles environments (`p45` and `p46`).

### 6.2. Baselines

We compare QGF, as well as its variant QGF-Jacobian (which includes the Jacobian from Eq. (8)), to a representative set of baselines from prior work. We group the baselines into two categories:

**(1) Test-time methods**, which use a policy pre-trained with BC and a learned critic to optimize sampled actions at test time: **BFN** [35, 19] samples $N$ actions from the actor and picks the highest value one according to the critic; **GradStep** improves fully denoised actions at test time by taking a few steps in the gradient direction [41, 65]; **QFQL** [24] guides each denoising step with the OOD gradient in Section 4 (*i.e.*, $\nabla_{a_t} Q(s, a_t)$); **BPTT**, a novel baseline inspired by DQL [63], uses the BPTT gradient estimator for guidance; **CFGRL** [14],

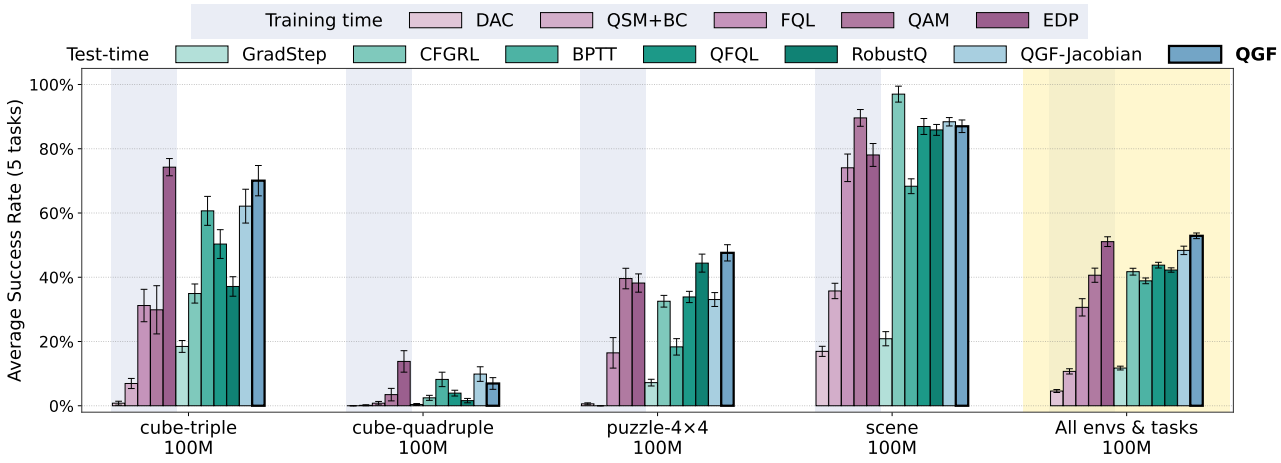


**Figure 5:** Offline RL performance at 500k training steps (20 tasks, 10 seeds): QGF beats all previous test-time methods and is competitive with the best training-time method. See task breakdown in Fig. 11.

which trains a policy by conditioning it on advantage values and samples with classifier free guidance at test time; **RobustQ**, a novel baseline inspired by robust classifiers [55, 26, 60, 22] in classifier guidance, takes gradients on a "robust" $Q$ function $Q(s, a_t, t)$ trained with noisy actions and denoising time steps as inputs.

**(2) Training-time methods**, which train the policy to maximize $Q(s, a)$ during training: **FQL** [50] learns a distilled one-step policy for ascending on the critic gradient; **EDP** [25] uses a first-order Euler approximation of the denoised action; **QAM** [33] uses adjoint matching to replace back propagating through the denoising process; **DAC** [13] trains a diffusion model to match base BC distribution tilted by $\exp(Q(s, a))$; **QSM+BC** add BC regularization to QSM [53], which uses a diffusion model to match the gradient of $Q(s, a)$.

See a more detailed description of these methods in Section I. To ensure a fair comparison of policy extraction methods, the same critic $Q(s, a)$ is used across all methods. We use the IQL [29] Q-value function described in the previous section. See Section 6.5 for details on different critic formulations.

### 6.3. Comparing QGF with Test-Time and Train-Time Baselines

Fig. 5 shows that QGF outperforms previous gradient-based guidance algorithms (QFQL, BPTT, RobustQ) significantly. This shows the strength of our proposed gradient estimator in Eq. (9). We speculate that this is due to its lower variance as shown in Fig. 3. QGF also significantly outperforms CFGRL and GradStep, indicating that gradient guidance during each denoising step is more effective than classifier free guidance, and than using the critic gradient to change the fully denoised action only. In addition, QGF also outperforms QGF-Jacobian, showing that not using the Jacobian in the gradient estimator improves performance, which is also confirmed later in Fig. 8 on harder OGBench tasks.

Compared to methods that explicitly train the policy to maximize Q functions (which we call training-time methods), QGF only trains the policy with a standard flow matching loss, and uses the Q function gradient to guide action sampling at test-time. Training-time methods typically require tuning a hyperparameter that balances reward maximization against a behavioral constraint (such as the weight for the behavioral constraint in EDP), and good performance usually requires extensive tuning of this hyperparameter. In comparison, QGF does not need to tune such a parameter during training. Instead, its behavior can be adjusted at test time without needing to re-train a policy. However, we still compare to representative training-time methods (FQL, EDP, QAM, DAC, QSM+BC). Surprisingly, QGF outperforms most training-time methods and is competitive and slightly better than the best. This demonstrates that test-time RL is a powerful policy optimization technique. We note that the best training-time baseline method, EDP, also uses a first-order approximation of the denoised action, showing the effectiveness of this approximation.

**Scaling test-time compute.** One effective way to improve policy performance when additional test-time compute is available is to incorporate best-of-N (BFN) sampling. In BFN, multiple samples are generated from a

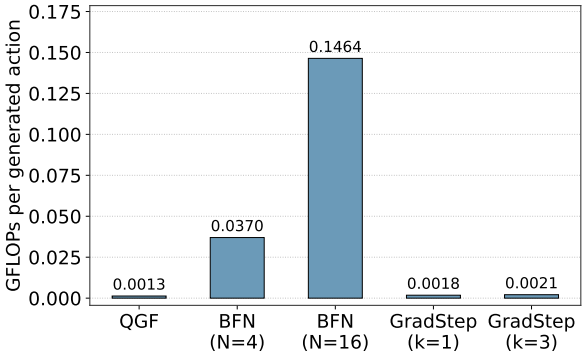


**Figure 6:** Compute requirements for different test-time methods: BFN needs orders of magnitude more FLOPs compared to QGF and GradStep.

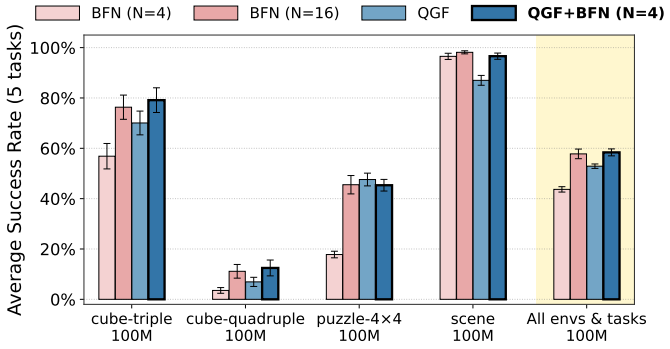


**Figure 7:** Offline RL performance at 500k training steps (20 tasks, 10 seeds) evaluated with best-of-N sampling. QGF outperforms BFN (N=4), and QGF+BFN matches BFN with much higher test-time compute budget. See task breakdown in Fig. 12.

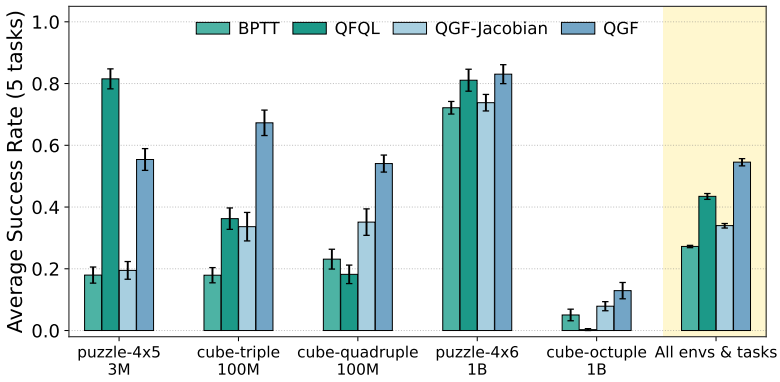


**Figure 8: Offline goal-conditioned RL performance** at 1M training steps (25 tasks, 10 seeds): While QGF underperforms the best baseline on the simplest task, it is consistently the best performing method for the harder tasks, showing that the gradient estimator scales well to long horizon tasks. See task breakdown in Fig. 13.

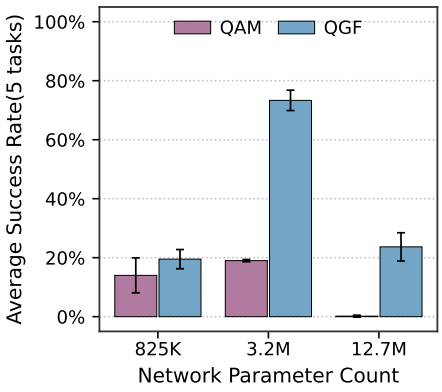


**Figure 9: Scaling model size of QGF**: compared to training-time methods such as QAM [33], QGF benefits much more from increasing the model size. Averaged over 5 tasks in `c3`.

reference policy and we select the sample with the highest Q-value. This has proven to be an effective policy extraction method for both robotic control [35, 19, 45] and language modeling [56]. However, BFN is very expensive since it requires rolling out the entire denoising process multiple times to select a promising action. In fact, Fig. 6 shows that BFN is orders of magnitude more expensive than QGF and other test-time methods that we consider. In Fig. 7, we compare QGF to methods that utilize best-of-N sampling. We introduce a variant of our method, QGF+BFN, which samples $N$ actions from QGF and picks the highest value one ranked by the critic. As we can see, while BFN (N=4) alone uses drastically more test-time compute than QGF, it actually performs worse, indicating that QGF is a more effective value maximizer on its own. With even more test-time compute, BFN (N=16) catches up, achieving similar and slightly better performance than QGF. However, when given more compute, QGF+BFN is able to beat QGF, and match BFN (N=16) with a smaller test-time compute budget, only requiring 4 samples instead of 16.

### *6.4. Scaling QGF to Harder Tasks and Larger Models*

To test whether our method can solve even harder tasks than those in Section 6.3, we also evaluate under the goal-conditioned RL setting instead of single-task RL. We evaluate QGF and its variant QGF-Jacobian with the two most relevant baselines that use critic gradients at test-time (BPTT, QFQL) on the five most challenging environments in OGBench. Since these are hard long horizon tasks, we use the corresponding larger datasets that support learning [49]. In this experiment, we train IQL value functions with DQC [34] since the tasks have

a very long horizon. Fig. 8 shows that on the easiest task, p45, QGF actually underperforms QFQL. However, as we move to harder tasks, QGF is consistently the best-performing method, showing that our critic gradient estimator in Eq. (9) scales well on hard long horizon tasks. It is also worth noting that on the hardest tasks, QGF consistently outperforms the QGF-Jacobian variant. This shows that it is beneficial to use a lower-variance gradient estimator that does not include the Jacobian.

To evaluate whether QGF scales well with model size, we compare QGF at different model sizes to a training-time baseline, QAM [33]. Fig. 9 shows the performance for the two agents using three model sizes for both the actor and the critic, and we find that when scaling the model size from 800k parameters to 3.2M parameters, QAM does not improve while QGF experiences a nearly 4× jump in performance. When we increase the model size past a certain threshold (e.g., 12.7M parameters), both methods experience overfitting, though QGF suffers less while QAM results in a policy that is unable to complete the task.

### 6.5. QGF with Different Critics

All the above results compare QGF and all baseline methods with an in-sample learning critic using IQL [29]. Here, we ask whether QGF can also work with critics trained with actions sampled from the policy via $Q$ bootstrapping [38, 16, 18] since these critics perform extremely well when tuned carefully [48]. We choose to train the critic with action sampling from QAM [33] because it is a top-performing method. Fig. 10 shows that QGF using the QAM-based $Q$ function performs much better than IQL-based QGF, and also performs better than QAM with $Q$ bootstrapping. This shows that QGF is an extremely effective policy extraction method that can work with different types of $Q$ functions.

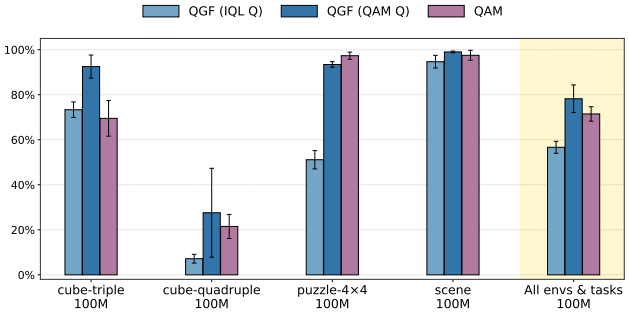


**Figure 10:** QGF can work with different types of critics and performs better when critics are better (20 tasks, 4 seeds). See task break down in Fig. 19.

## 7. Conclusion

We present QGF, a test-time RL algorithm for improving flow policies trained with behavioral cloning. By guiding each denoising step with a critic gradient evaluated at an approximated clean action, QGF avoids unreliable gradients at noisy actions as well as the cost and instability of backpropagating through the full denoising process. Empirically, QGF outperforms prior test-time guidance methods, is competitive with strong training-time RL baselines, and scales favorably to harder tasks and larger models. These results suggest that test-time gradient guidance is a practical and scalable alternative to actor-critic policy optimization for expressive generative policies in continuous control.

## 8. Acknowledgements

This research was partly supported by AFOSR FA9550-22-1-0273 and DARPA ANSR. This research used the Savio computational cluster resource provided by the Berkeley Research Computing program at UC Berkeley. We would like to thank Seohong Park, Chung Min Kim, and Kai Nelson for helpful discussions and feedback on early drafts of the paper.

## 9. Author Contributions

ZZ led the project, carried out the major experiments, and led the writing. AP ran substantial experiments that helped iterate and analyze the method, and also assisted with writing. CX contributed to early project ideation and discussions, and helped with experiments, method analysis, and writing. QL ran the goal-conditioned RL experiments and, together with TS, KF, and SL, provided valuable guidance and advice throughout the project and contributed to writing.

## References


[1] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations*, 2018.

[2] Gaon An, Seungyong Moon, Jang-Hyun Kim, and Hyun Oh Song. Uncertainty-based offline reinforcement learning with diversified q-ensemble. *Advances in neural information processing systems*, 34:7436–7447, 2021.

[3] Huayu Chen, Cheng Lu, Chengyang Ying, Hang Su, and Jun Zhu. Offline reinforcement learning via high-fidelity generative behavior modeling. *arXiv preprint arXiv:2209.14548*, 2022.

[4] Tianyi Chen, Haitong Ma, Na Li, Kai Wang, and Bo Dai. One-step flow policy mirror descent. *arXiv preprint arXiv:2507.23675*, 2025.

[5] Tianyu Chen, Zhendong Wang, and Mingyuan Zhou. Diffusion policies creating a trust region for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 37:50098–50125, 2024.

[6] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. *Advances in neural information processing systems*, 5, 1992.

[7] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

[8] Shutong Ding, Ke Hu, Zhenhao Zhang, Kan Ren, Weinan Zhang, Jingyi Yu, Jingya Wang, and Ye Shi. Diffusion-based reinforcement learning via q-weighted variational policy optimization. *Advances in Neural Information Processing Systems*, 37:53945–53968, 2024.

[9] Zihan Ding and Chi Jin. Consistency models as a rich and efficient policy class for reinforcement learning. *arXiv preprint arXiv:2309.16984*, 2023.

[10] Carles Domingo-Enrich, Michal Drozdzal, Brian Karrer, and Ricky T. Q. Chen. Adjoint matching: Fine-tuning flow and diffusion generative models with memoryless stochastic optimal control, 2025.

[11] Nicolas Espinosa-Dice, Yiyi Zhang, Yiding Chen, Bradley Guo, Owen Oertell, Gokul Swamy, Kiante Brantley, and Wen Sun. Scaling offline rl via efficient and expressive shortcut models. *arXiv preprint arXiv:2505.22866*, 2025.

[12] Jiajun Fan, Shuaike Shen, Chaoran Cheng, Yuxin Chen, Chumeng Liang, and Ge Liu. Online reward-weighted fine-tuning of flow matching with wasserstein regularization. In *The Thirteenth International Conference on Learning Representations*, 2025.

[13] Linjiajie Fang, Ruoxue Liu, Jing Zhang, Wenjia Wang, and Bing-Yi Jing. Diffusion actor-critic: Formulating constrained policy iteration as diffusion noise regression for offline reinforcement learning. *arXiv preprint arXiv:2405.20555*, 2024.

[14] Kevin Frans, Seohong Park, Pieter Abbeel, and Sergey Levine. Diffusion guidance is a controllable policy improvement operator. *arXiv preprint arXiv:2505.23458*, 2025.

[15] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

[16] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[17] Divyansh Garg, Joey Hejna, Matthieu Geist, and Stefano Ermon. Extreme q-learning: Maxent rl without entropy. *arXiv preprint arXiv:2301.02328*, 2023.

[18] 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*, pages 1861–1870. PMLR, 2018.

[19] 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. *arXiv preprint arXiv:2304.10573*, 2023.

[20] Longxiang He, Li Shen, Linrui Zhang, Junbo Tan, and Xueqian Wang. Diffcps: Diffusion model based constrained policy search for offline reinforcement learning. *arXiv preprint arXiv:2310.05333*, 2023.

[21] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[22] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *Advances in neural information processing systems*, 32, 2019.

[23] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *International conference on machine learning*, pages 1627–1635. PMLR, 2017.

[24] Yejun Jang, Hong Chul Nam, Jeong Min Park, Gimin Bae, and Hyun Kwon. Q-guided flow q-learning. In *CoRL 2025 Workshop RemembeRL*.

[25] Bingyi Kang, Xiao Ma, Chao Du, Tianyu Pang, and Shuicheng Yan. Efficient diffusion policies for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 36:67195–67212, 2023.

[26] Bahjat Kawar, Roy Ganz, and Michael Elad. Enhancing diffusion-based image synthesis with robust classifier guidance. *arXiv preprint arXiv:2208.08664*, 2022.

[27] Diederik Kingma and Ruiqi Gao. Understanding diffusion objectives as the elbo with simple data augmentation. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 65484–65516, 2023.

[28] Prajwal Koirala and Cody Fleming. Flow-based single-step completion for efficient and expressive policy learning. *arXiv preprint arXiv:2506.21427*, 2025.

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

[30] Aviral Kumar, Xue Bin Peng, and Sergey Levine. Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*, 2019.

[31] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[32] Julien Launay, Iacopo Poli, François Boniface, and Florent Krzakala. Direct feedback alignment scales to modern deep learning tasks and architectures. *Advances in neural information processing systems*, 33:9346–9360, 2020.

[33] Qiyang Li and Sergey Levine. Q-learning with adjoint matching. *arXiv preprint arXiv:2601.14234*, 2026.

[34] Qiyang Li, Seohong Park, and Sergey Levine. Decoupled q-chunking. *arXiv preprint arXiv:2512.10926*, 2025.

[35] Qiyang Li, Zhiyuan Zhou, and Sergey Levine. Reinforcement learning with action chunking. *arXiv preprint arXiv:2507.07969*, 2025.

[36] Zechu Li, Rickmer Krohn, Tao Chen, Anurag Ajay, Pulkit Agrawal, and Georgia Chalvatzaki. Learning multimodal behaviors from scratch with diffusion policy gradient. *Advances in Neural Information Processing Systems*, 37:38456–38479, 2024.

[37] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random feedback weights support learning in deep neural networks. *arXiv preprint arXiv:1411.0247*, 2014.

[38] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[39] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

[40] Lei Lv, Yunfei Li, Yu Luo, Fuchun Sun, Tao Kong, Jiafeng Xu, and Xiao Ma. Flow-based policy for online reinforcement learning. *arXiv preprint arXiv:2506.12811*, 2025.

[41] Max Sobol Mark, Tian Gao, Georgia Gabriela Sampaio, Mohan Kumar Srirama, Archit Sharma, Chelsea Finn, and Aviral Kumar. Policy agnostic rl: Offline rl and online rl fine-tuning of any class and backbone. *arXiv preprint arXiv:2412.06685*, 2024.

[42] David McAllister, Miika Aittala, Tero Karras, Janne Hellsten, Angjoo Kanazawa, Timo Aila, and Samuli Laine. Finite difference flow optimization for rl post-training of text-to-image models. *arXiv preprint arXiv:2603.12893*, 2026.

[43] David McAllister, Songwei Ge, Brent Yi, Chung Min Kim, Ethan Weber, Hongsuk Choi, Haiwen Feng, and Angjoo Kanazawa. Flow matching policy gradients. *arXiv preprint arXiv:2507.21053*, 2025.

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

[45] Mitsuhiko Nakamoto, Oier Mees, Aviral Kumar, and Sergey Levine. Steering your generalists: Improving robotic foundation models via value guidance. *arXiv preprint arXiv:2410.13816*, 2024.

[46] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

[47] Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. *arXiv preprint arXiv:2410.20092*, 2024.

[48] Seohong Park, Kevin Frans, Sergey Levine, and Aviral Kumar. Is value learning really the main bottleneck in offline rl? *Advances in Neural Information Processing Systems*, 37:79029–79056, 2024.

[49] Seohong Park, Kevin Frans, Deepinder Mann, Benjamin Eysenbach, Aviral Kumar, and Sergey Levine. Horizon reduction makes rl scalable. *arXiv preprint arXiv:2506.04168*, 2025.

[50] Seohong Park, Qiyang Li, and Sergey Levine. Flow Q-learning. *arXiv preprint arXiv:2502.02538*, 2025.

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

[52] Samuel Pfrommer, Yixiao Huang, and Somayeh Sojoudi. Reinforcement learning for flow-matching policies. *arXiv preprint arXiv:2507.15073*, 2025.

[53] Michael Psenka, Alejandro Escontrela, Pieter Abbeel, and Yi Ma. Learning a diffusion model policy from rewards via q-score matching. *arXiv preprint arXiv:2312.11752*, 2023.

[54] Allen Z Ren, Justin Lidard, Lars L Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion policy policy optimization. *arXiv preprint arXiv:2409.00588*, 2024.

[55] Shibani Santurkar, Andrew Ilyas, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Image synthesis with a single (robust) classifier. *Advances in neural information processing systems*, 32, 2019.

[56] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[57] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021, 2020.

[58] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[59] Denis Tarasov, Vladislav Kurenkov, Alexander Nikulin, and Sergey Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[60] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, 2018.

[61] John Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. *Advances in neural information processing systems*, 9, 1996.

[62] Yinuo Wang, Likun Wang, Yuxuan Jiang, Wenjun Zou, Tong Liu, Xujie Song, Wenxuan Wang, Liming Xiao, Jiang Wu, Jingliang Duan, et al. Diffusion actor-critic with entropy regulator. *Advances in Neural Information Processing Systems*, 37:54183–54204, 2024.

[63] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022.

[64] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[65] Haoran Xu, Kaiwen Hu, Somayeh Sojoudi, and Amy Zhang. Reinforcement learning via value gradient flow. In *The Fourteenth International Conference on Learning Representations*.

[66] Haoran Xu, Li Jiang, Jianxiong Li, Zhuoran Yang, Zhaoran Wang, Victor Wai Kin Chan, and Xianyuan Zhan. Offline rl with no ood actions: In-sample learning via implicit value regularization. *arXiv preprint arXiv:2303.15810*, 2023.

[67] Shuchen Xue, Chongjian Ge, Shilong Zhang, Yichen Li, and Zhi-Ming Ma. Advantage weighted matching: Aligning rl with pretraining in diffusion models. *arXiv preprint arXiv:2509.25050*, 2025.

[68] Long Yang, Zhixiong Huang, Fenghao Lei, Yucun Zhong, Yiming Yang, Cong Fang, Shiting Wen, Binbin Zhou, and Zhouchen Lin. Policy representation via diffusion probability model for reinforcement learning. *arXiv preprint arXiv:2305.13122*, 2023.

[69] Ruoqi Zhang, Ziwei Luo, Jens Sjölund, Thomas B Schön, and Per Mattsson. Entropy-regularized diffusion policy with q-ensembles for offline reinforcement learning. *Advances in neural information processing systems*, 37:98871–98897, 2024.

[70] Shiyuan Zhang, Weitong Zhang, and Quanquan Gu. Energy-weighted flow matching for offline reinforcement learning. *arXiv preprint arXiv:2503.04975*, 2025.

## A. Result Details

All performance result plots in the main paper and in the appendix report 95% confidence interval as the error bar. Fig. 12 and Fig. 11 contain the performance of QGF and baselines for individual tasks for each of the singletask OGBench domains. Fig. 13 contain the performance of individual tasks for the goal-conditioned OGBench domains.

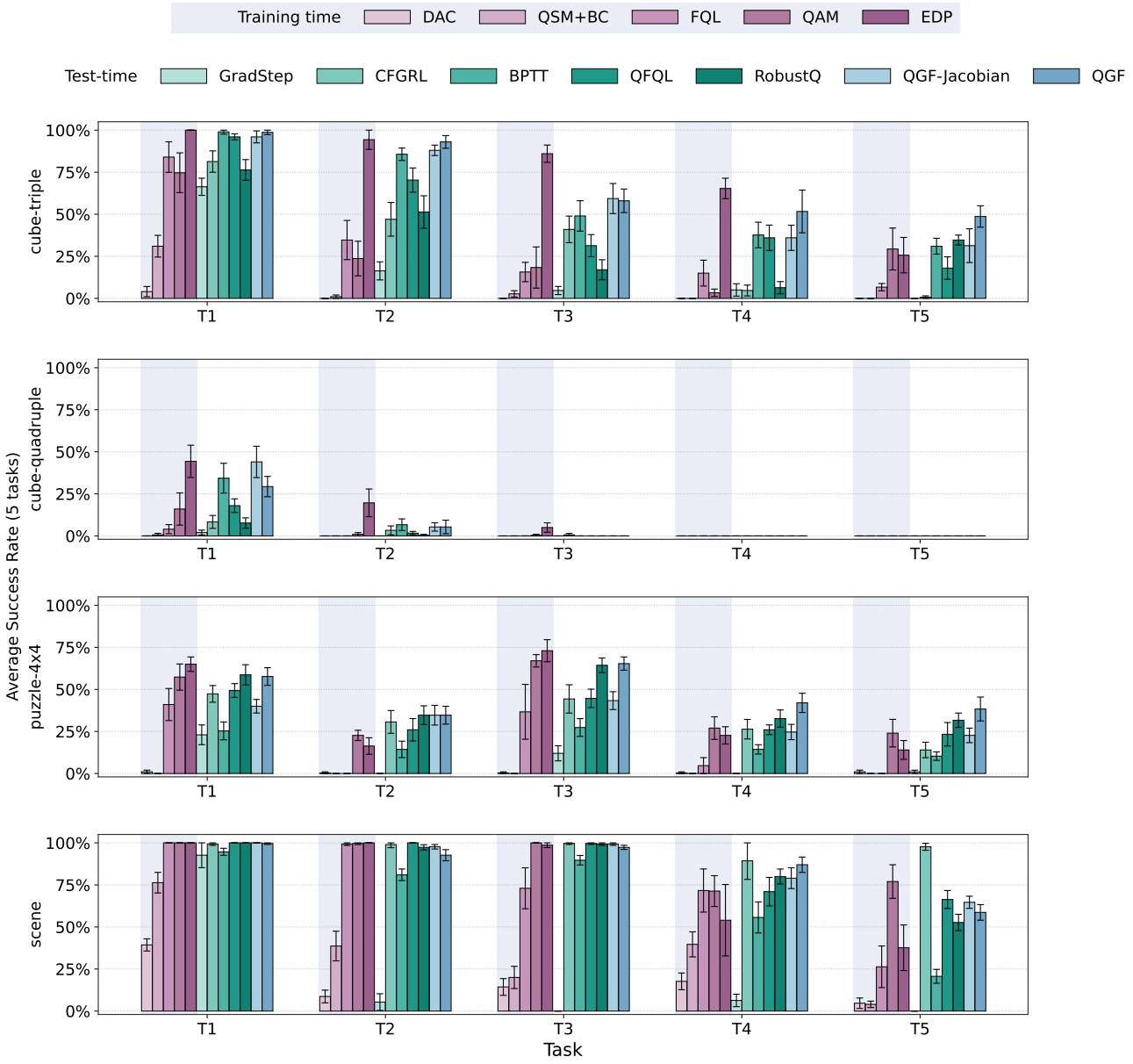


**Figure 11:** Offline RL performance at 500k training steps per environment (10 seeds each).

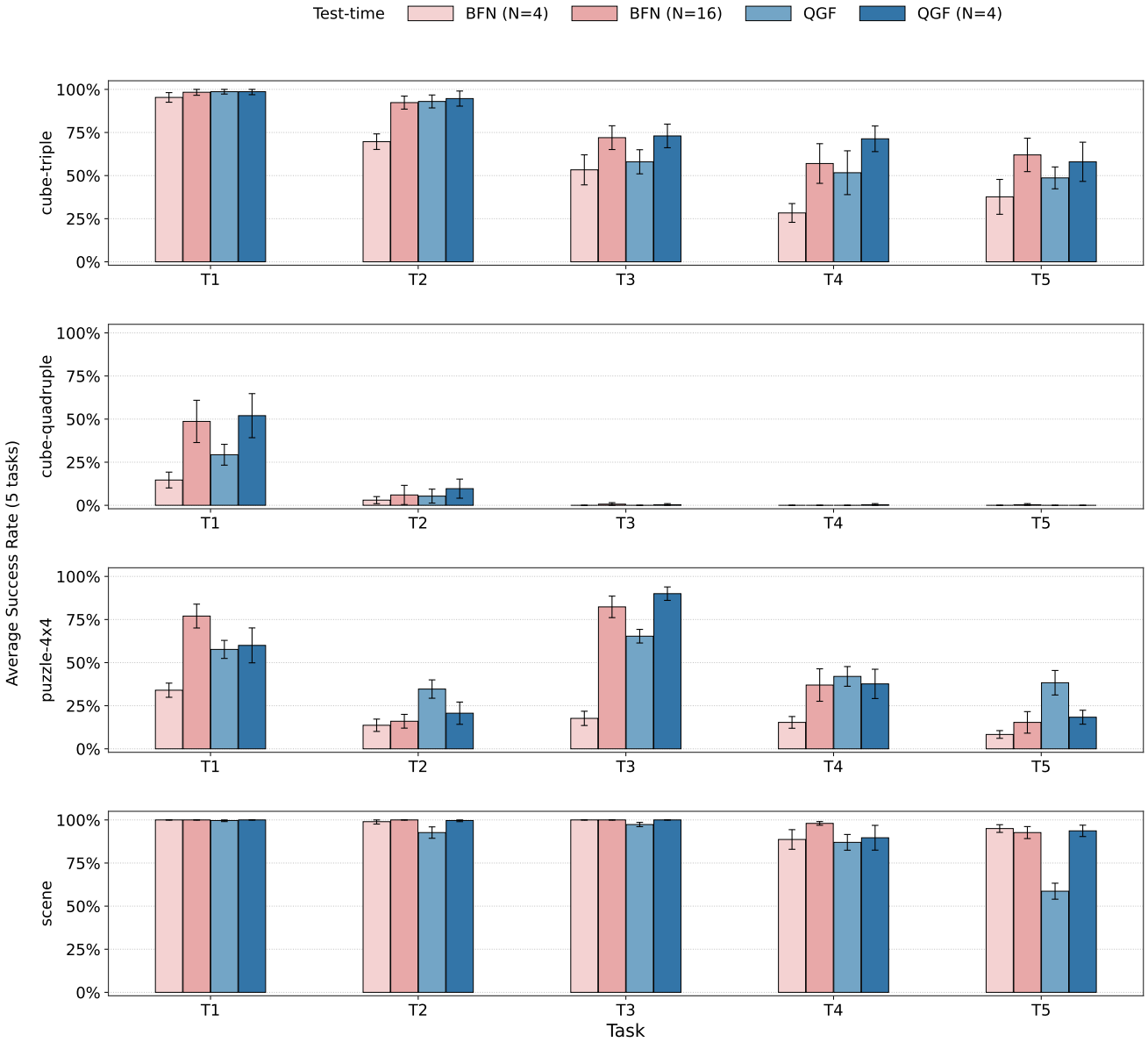


**Figure 12:** Offline RL with Best-of-N sampling at 500k training steps per environment (10 seeds each).

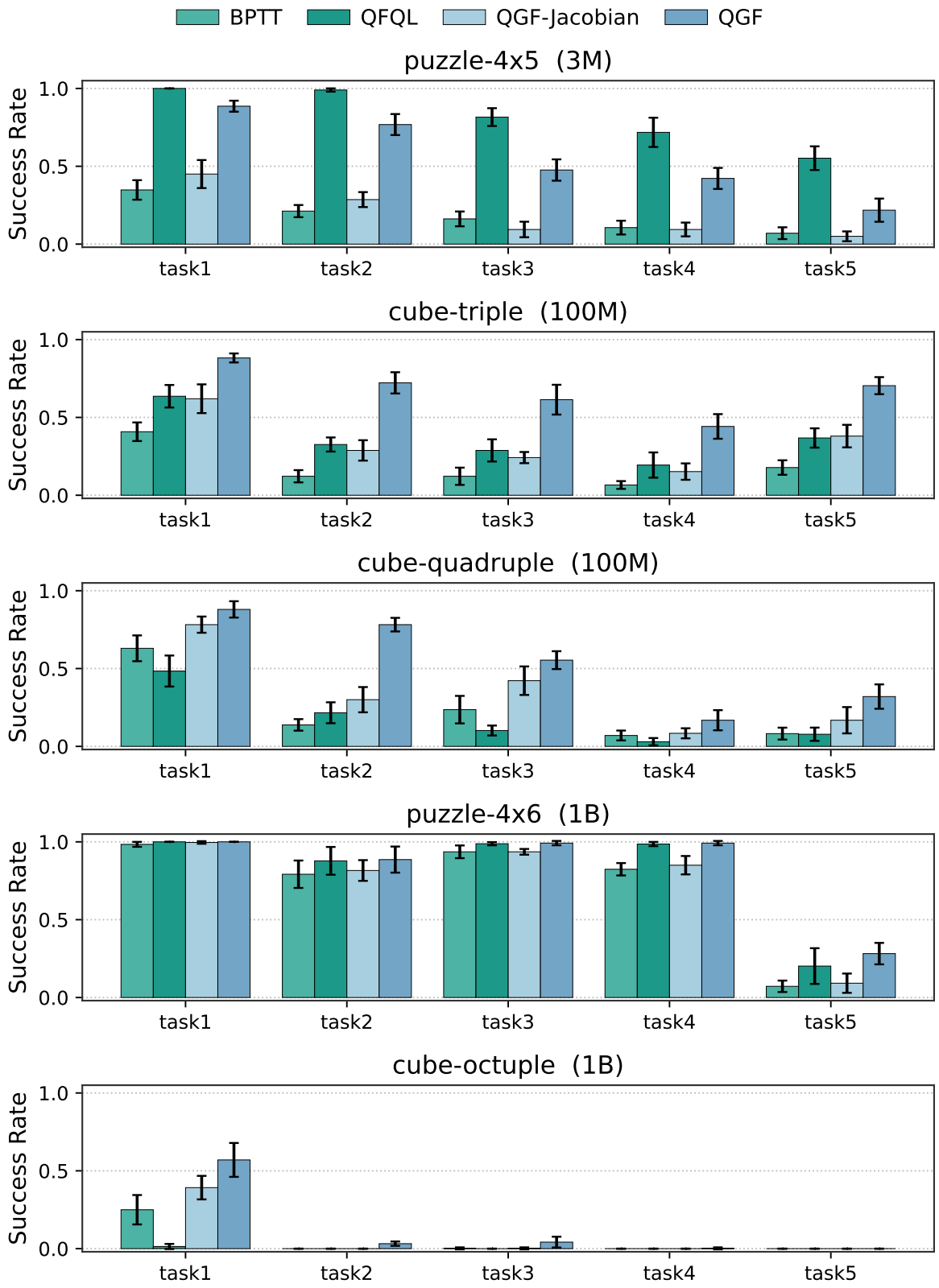


**Figure 13:** Offline goal-conditioned RL at 1M training steps (10 seeds each). See the domain-specific hyperparameters in Table 4.

## B. Extra Analysis on BPTT Gradient Guidance

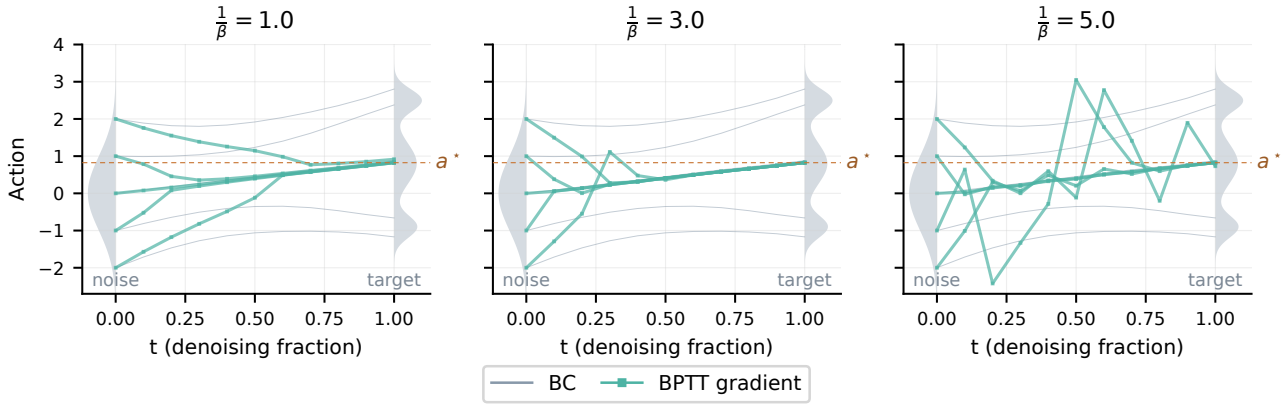


**Figure 14:** We find that the BPTT gradient can be unstable for higher guidance weights and certain target distributions.

Here we offer another example of our 1-D illustrative denoising example described in Section 4. In this example, we find that the BPTT gradient can be extremely unstable, as shown in Fig. 14. This suggests that using such gradient guidance could lead to suboptimal results, as confirmed by our experiments in Fig. 5 and Fig. 8.

## C. QGF Variants

We present in Fig. 15 the comparison between QGF and many variants of QGF. See the task specific breakdown in Fig. 16. QGF, QGF-Jacobian, and QGF-chain are explained in Section 5, and we detail the other variants below:

**QGF Distill / QGF-Jacobian Distill.** Instead of using the first-order approximation in Eq. (7) to approximate the fully denoised action, one can also train a "distilled" velocity field to denoise noisy actions. For any noisy action $a_t$ at timestep $t$, we can train a velocity field $v_d(s, a_t, t)$ with loss:

$$\|v_d(s, a_t, t) - \text{ODE}(a_t)\|^2,$$

where $\text{ODE}(a_t)$ represent the full denoising chain of the reference policy ($\text{ODE}(a_t) = a_t + \int_t^1 v_\theta(a_\tau, \tau) d\tau$). Then, we approximate the clean action with $\hat{a}_1 = v_d(s, a_t, t)$, and approximate the gradient with

$$\nabla_{a_t} Q(s, \hat{a}_1) = J^T \nabla_{\hat{a}_1} Q(s, \hat{a}_1).$$

QGF-Jacobian Distill would calculate the Jacobian $J = \frac{\partial \hat{a}_1}{\partial a_t}$ by taking derivative through $v_d$; QGF Distill would simply set $J \approx \hat{J} = I$.

**QGF-Jacobian Smooth.** Since we found in Fig. 3 that QGF-Jacobian has higher "variance" than QGF, we explore whether we can reduce the variance of the QGF-Jacobian estimator with some "smoothing" method. Here we also use the one-euler-step approximation of $\hat{a}_1 = a_t + (1-t)v_\theta(s, a_t, t)$, but replace the single-point Jacobian $J = \frac{\partial \hat{a}_1}{\partial a_t}$ with a Monte Carlo average over $K$ Gaussian perturbations of $a_t$:

$$\hat{J}(a_t) = \frac{1}{K} \sum_{k=1}^{K} \frac{\partial \hat{a}_1}{\partial a_t}\bigg|_{a_t + \epsilon_k}, \qquad \epsilon_k \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, , \sigma^2 I)$$

The rest of the chain rule is unchanged. The full Q-gradient is:

$$\nabla_{a_t} Q = \hat{J}(a_t)^\top \nabla_{\hat{a}_1} Q(s, \hat{a}_1)$$

**QGF Regularized.** We investigate an estimator where the Jacobian is regularized towards the identity:

$$\nabla_{a_t} Q(s, \hat{a}_1) = \nabla_{\hat{a}_1} Q(s, \hat{a}_1) \cdot (I + \epsilon \cdot \frac{\partial \hat{a}_1}{\partial a_t}) \tag{10}$$

where $\epsilon$ is some small value. We use $\epsilon = 0.1$ in Fig. 15.

**QGF-Jacobian Ortho** This is another regularization method for QGF-Jacobian that orthogonalizes the Jacobian matrix. We orthogonalize with Singular Value Decomposition (SVD): $J = U\Sigma V^\top$, then

$$J_{\text{ortho}} = UV^\top.$$

This is the nearest orthogonal matrix to $J$ (in Frobenius norm). The gradient then becomes:

$$\nabla_{a_t} Q(s, \hat{a}_1) = J_{\text{ortho}}^\top \nabla_{\hat{a}_1} Q(s, \hat{a}_1).$$

Intuitively, $J$ rotates and scales; orthogonalization keeps the rotation but discards the scaling, so the guidance direction is preserved but its magnitude is no longer amplified or shrunk by the singular values of the denoising map.

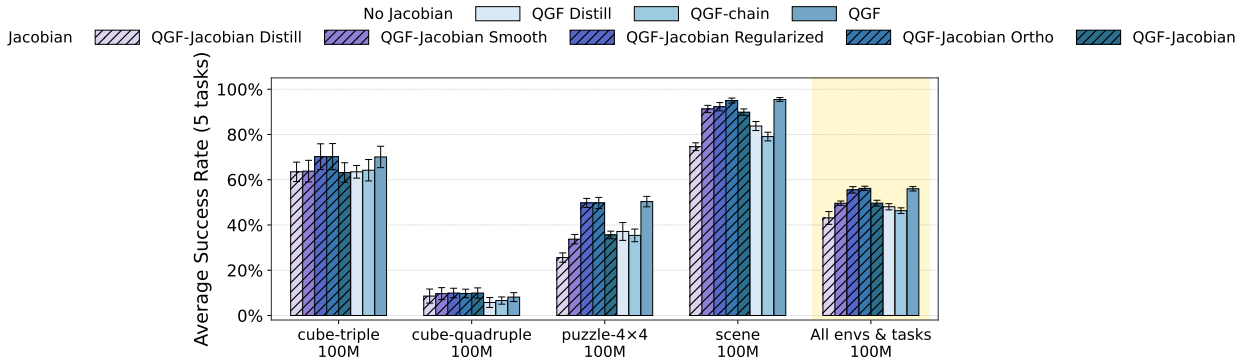


**Figure 15:** Offline RL performance at 500k training steps (20 tasks, 10 seeds) of different QGF variants. Methods that apply the Jacobian are shaded. See task specific break down in Fig. 16.

Fig. 15 shows the offline RL performance of all the QGF variants on OGBench. Surprisingly, both QGF-Jacobian Regularized and QGF-Jacobian Ortho actually performs similarly to QGF, and all three perform better than other variants with the Jacobian. The further shows that including the Jacobian actually hurts performance (likely due to the high variance of the gradient estimator). However, if one regularized the Jacobian, the performance can be recovered. That said, the simplest method remains just setting $J \approx \hat{J} = I$. Surprisingly, QGF Distill actually also underperforms QGF, and performs similarly to QGF-chain. We hypothesize that it also suffers from the same constraint that QGF-chain does, and places too much restrictions on strictly adhering to the dataset distribution.

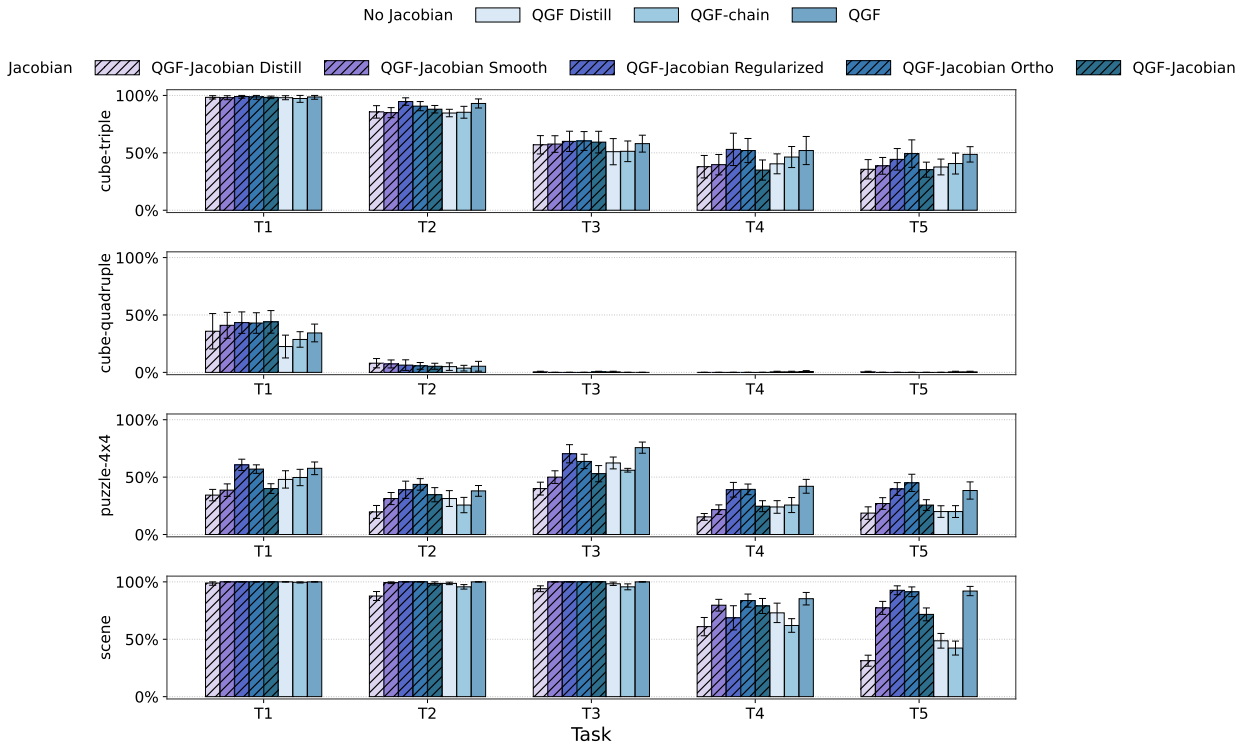


**Figure 16:** Full results for Fig. 15: Offline RL performance at 500k training steps (20 tasks, 10 seeds).

## D. OOD Gradient Exploits $Q$ function

Fig. 4 shows that QGF is an extremely effective optimizer for $Q$-values and does better than QGF-Jacobian, QGF-chain, and BPTT. It performs similarly to the oracle, which we take to be the best-of-n (BFN) sampling agent. The BFN agent is known to be extremely effective at optimizing actions to maximize $Q$-values [35, 19], which aligns with our results in Fig. 7. Curiously, the OOD gradient is an even more effective optimizer than BFN, leading to even higher $Q$-values. However, Fig. 5 shows that the QFQL agent, which uses the OOD gradient, does not perform well in practice. This implies that the OOD gradient may be exploiting the $Q$ function by generating actions outside the distribution of the dataset (which we will call OOD actions) since the gradient itself is ill formed (see Section 4). This may not come as a surprise since it is well-known that critics trained with offline RL can over-estimate $Q$-values of OOD actions [31, 29, 15]. To investigate whether the actions from the OOD gradient are out of distribution, we plot both its distance to the oracle BFN action and its nearest-neighbor distance to actions from the offline dataset. Fig. 17 shows that actions from the OOD gradient has the largest distance on both distance metrics out of all the gradient estimators, showing that it produces actions least similar to dataset actions and optimal actions. This shows that the OOD gradient indeed denoised to some OOD actions that exploits the $Q$-values, and so such actions does not actually lead to good performance.

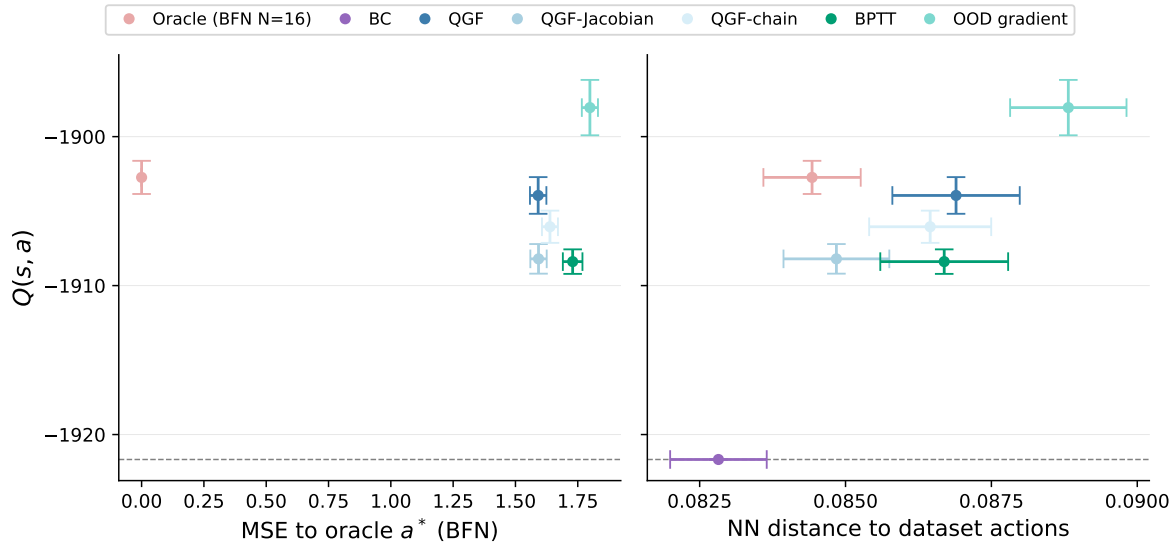


**Figure 17:** The OOD gradient leads to a denoised action with higher $Q$ value than all other methods by exploiting the critic on OOD actions. **(left)**: the OOD gradient leads to actions that are farthest away from the BFN oracle actions. **(right)**: the OOD gradient leads to actions that has the largest nearest-neighbor (NN) distance to dataset actions. Aggregated over 20 tasks, 256 observations, 4 seeds.

## E. Sensitivity of Different Gradient Estimators to Noise

Fig. 3 shows the sensitivity of gradient estimators to noise in terms of cosine similarity between $G(s, a_t)$ and $G(s, a_t + \epsilon)$. That plot averages over 20 different tasks in OGBench, and Fig. 18 shows the task-specific results.

## F. Critic Training

We follow the action chunking setting [35] and learn both actors that produces action chunks and critics that are conditioned on action chunks (*i.e.*, $a_{1:h}$ for chunk size $h$).

**Singletask IQL critic.** Our method is agnostic to the specific critic training recipe used. In practice we chose to use IQL [29] as the main critic training method because it can be trained using only in-sample learning and we can completely separate actor and critic training. All of the different methods in Fig. 5 are evaluated using an IQL Q function, which is learned as follows:

For an offline dataset $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}$, we train a critic $Q_\theta$ and value function $V_\psi$ using the following objectives:

$$\mathcal{L}_Q(\theta) = \mathbb{E}_{(s_t, a_{t:t+h}, r_{t:t+h}, s_{t+h})\sim\mathcal{D}}\left[\left(Q_\theta(s_t, a_{t:t+h}) - \left(\sum_{t'=t}^{t+h-1} [\gamma^{t'-t} r'_t] + \gamma^h V_{\bar{\psi}}(s_{t+1})\right)^2\right], \tag{11}$$

$$\mathcal{L}_V(\psi) = \mathbb{E}_{(s_t, a_{t:t+h})\sim\mathcal{D}}\left[L_2^\tau\big(Q_{\bar{\theta}}(s_t, a_{t:t+h}) - V_\psi(s_t)\big)\right], \tag{12}$$

where $L_2^\tau(u) = |\tau - \mathbf{1}(u < 0)|u^2$ is the expectile regression loss (expectile $\tau$), and $\bar{\theta}$, $\bar{\psi}$ denote target/stop-gradient parameters. We train all IQL critics with an ensemble of 2 critics and aggregate them by taking the minimum over the ensemble.

**Goal-conditioned IQL critic.** We train goal-conditioned IQL critics following the critic training procedure in Decoupled Q Chunking (DQC) [34]. DQC decouples the chunk horizon for the policy ($h_a$) and the critic ($h_c$). The chunk horizon used for the critic is usually much larger to speed up value propagation. We train a total of 2 critics, $Q_c$ and $Q_a$, which has chunk horizon $h_c$ and $h_a$ respectively.

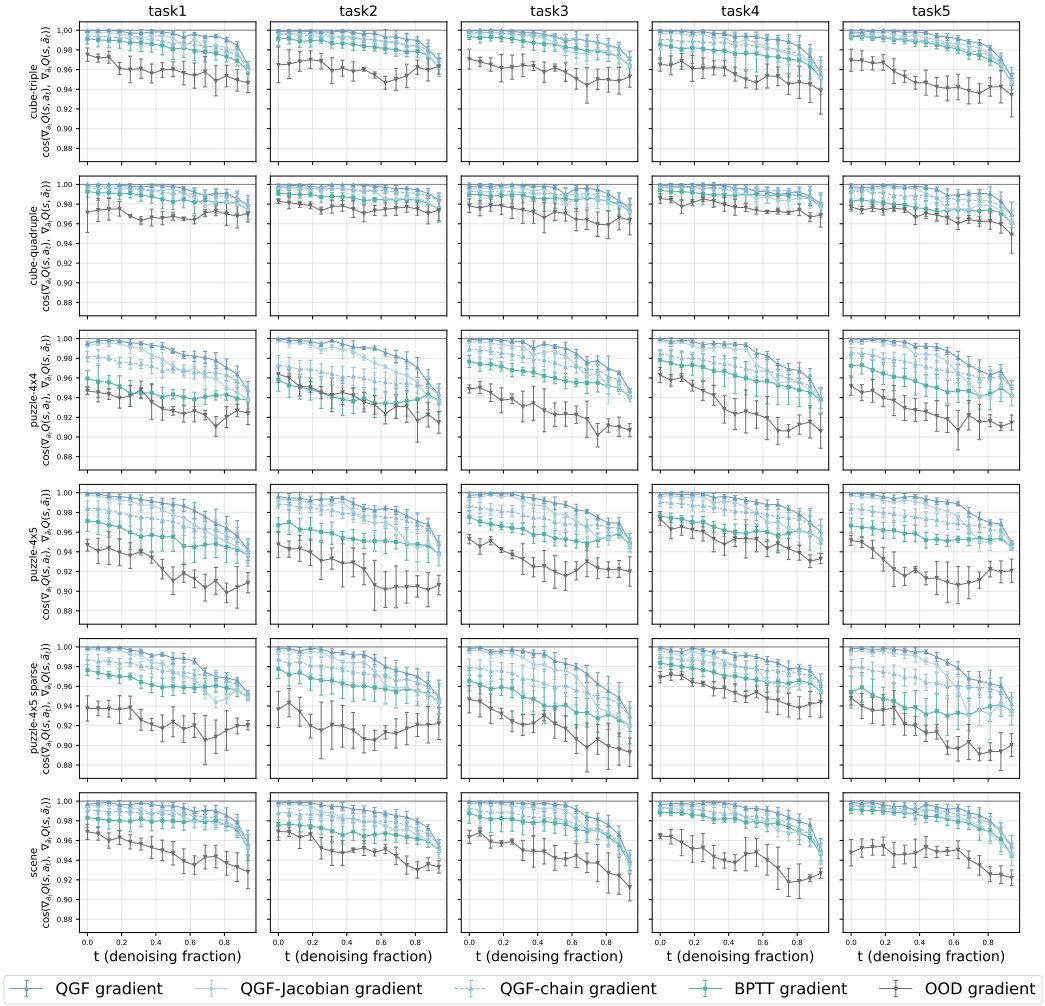


**Figure 18:** Full results for Fig. 3 on sensitivity of gradient estimator to noise for all 20 OGBench environments. 4 seeds each.

The long-horizon critic $Q_c$ is trained using the same IQL objective as above:

$$\mathcal{L}Q_c(\theta_c) = \mathbb{E}_{(s_t, a_{t:t+h_c}, r_{t:t+h_c}, s_{t+h_c})\sim\mathcal{D}}\left[\left(Q_c(s_t, g, a_{t:t+h_c}) - \left(\sum_{t'=t}^{t+h_c-1} \gamma^{t'-t} r_{t'} + \gamma^{h_c} V_{\bar{\psi}}(s_{t+h_c}, g)\right)\right)^2\right], \tag{13}$$

with the corresponding expectile value objective

$$\mathcal{L}_V(\psi)\mathbb{E}_{(s_t, g, a_{t:t+h_c})\sim\mathcal{D}}\left[L_2^{\tau}\big(Q_{\bar{\theta}c}(s_t, g, a_{t:t+h_c}) V_\psi(s_t, g)\big)\right]. \tag{14}$$

The short-horizon critic $Q_a$ is trained by distilling the optimistic value of extending a partial action chunk into a complete chunk. Concretely, for a partial chunk $a_{t:t+h_a}$, we regress $Q_a$ towards an upper expectile of the full-chunk critic values:

$$\mathcal{L}_{Q_a}(\theta_a)\mathbb{E}_{(s_t, g, a_{t:t+h_c})\sim\mathcal{D}}\left[L_2^{\kappa_d}\Big(Q_c(s_t, g, a_{t:t+h_c}) - Q_a(s_t, g, a_{t:t+h_a})\Big)\right], \tag{15}$$

where $\kappa_d > 0.5$ is the distillation expectile. This objective causes $Q_a$ to approximate the value of the best completion of the partial chunk under $Q_c$:

$$Q_a(s_t, g, a_{t:t+h_a}) \approx \max_{a_{t+h_a:t+h_c}} Q_c(s_t, g, [a_{t:t+h_a}, a_{t+h_a:t+h_c}]). \tag{16}$$

For policy extraction, we optimize and evaluate policies using only $Q_a$. This allows the policy to operate with a short execution horizon $h_a$ while still benefiting from the faster value propagation provided by the long-horizon critic $Q_c$.

## G. QGF with Other Types of Value Functions

In Section 6.5, we train the critic by minimizing the temporal difference (TD) error:

$$\mathbb{E}_{s,a,s' \in \mathcal{D}}(Q(s,a) - r(s,a) - \gamma \mathbb{E}_{\hat{a}_1 \sim \pi_{\text{QAM}}}[Q_{\bar{\phi}}(s', \hat{a}_1)])^2,$$

where we bootstrap to a $Q$ function evaluated at a sampled action from the QAM policy $\pi_{\text{QAM}}$.

To get the best critic possible, we following the original QAM paper [33] and train an ensemble of 10 $Q$ functions, and aggregate them by taking the mean of the ensemble values.

The full result is presented in Fig. 19. To avoid confusion, the QAM agent in Fig. 10 and Fig. 19 is trained using the critic described above by minimizing TD error, while the QAM agent in Fig. 5 uses IQL critic for comparability against other agents.

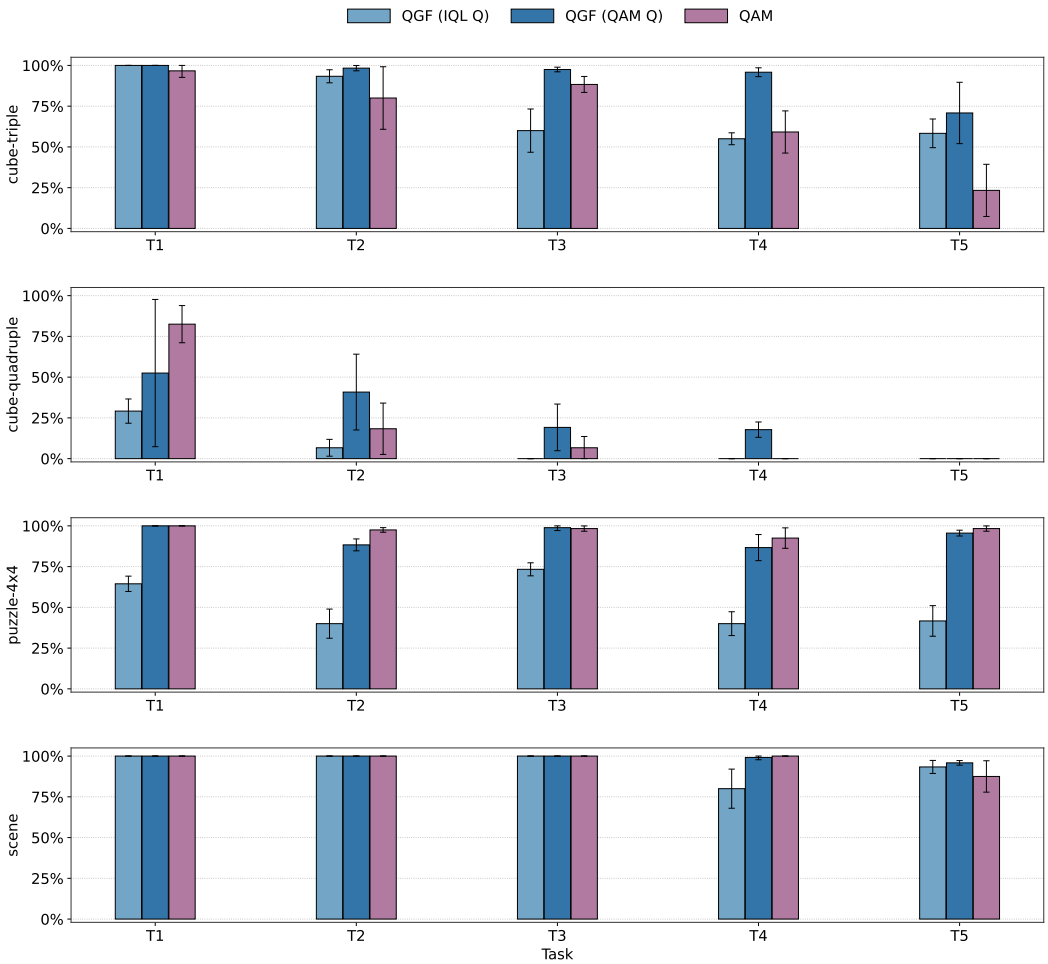


**Figure 19:** Full Results for Fig. 10: QGF with a QAM-based critic performs even better than QGF with an IQL-based critic on OGBench.

## H. Guidance Sensitivity Analysis

We perform a sensitivity analysis of the guidance weight $\frac{1}{\beta}$ in Eq. (5) for QGF in Fig. 20. We can see that for most environments, increasing guidance weight drastically improves policy performance. However, an overly large guidance weight can also hurt performance since it can push the actions outside of the dataset support.

## I. Details On Baseline Methods

**(1) Test-time methods**

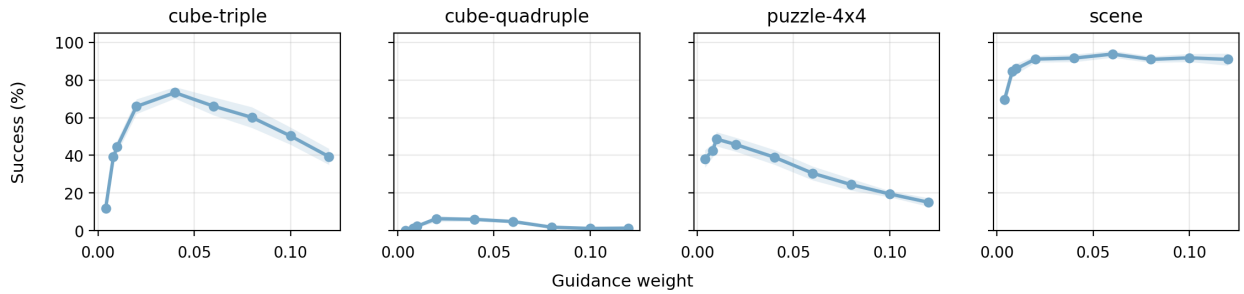


**Figure 20:** QGF performance against guidance weight: increasing guidance weight can drastically improve policy performance, but overly large weight can also hurt performance by pushing actions off manifold.

**Best-of-N (BFN)** performs rejection sampling on the base behavior cloning policy $\hat{\pi}$ trained with the flow matching objective. Specifically, the output action $a$ is selected as:

$$a \leftarrow \arg\max_{a_1,\cdots,a_N \sim \hat{\pi}} Q(s, a_t). \tag{17}$$

**BPTT** is a baseline we consider inspired by DQL [63] that computes guidance by backpropagating the critic objective through the *entire* denoising chain (i.e., full Backpropogation through time), yielding $\nabla_{a_t} Q(s, a_1)$ where $a_1$ is the action produced after all denoising steps, and $a_t$ is the partially-denoised action at timestep $t$. We compute the denoised action as

$$a_1 = \text{ODE}(a_t) = D_\theta^{(1)} \circ \cdots \circ D_\theta^{(t+\delta)} \circ D_\theta^{(t)}(s, a_t),$$

where $D_\theta^{(t)}$ denotes the $t$-th denoising step of the policy. Then we can write the gradient as:

$$\nabla_{a_t} Q(s,\ D_\theta^{(1)} \circ \cdots \circ D_\theta^{(t+\delta)} \circ D_\theta^{(t)}(s, a_t)), \tag{18}$$

and taking the gradient at $a_t$ requires differentiating through the chain of denoising steps $D_\theta^{(t)} \to \cdots \to D_\theta^{(1)}$.

**GradStep** is a baseline we consider inspired by Policy Agnostic RL (PA-RL) [41] that first samples a denoised action by running the full denoising process of our behavior cloning flow policy $\hat{\pi}$, then performs policy improvement by iteratively taking $L$ gradient ascent steps on the fully denoised action:

**Algorithm 2** GradStep

**Input:** $L$, $\alpha_{\text{step}}$, $\hat{\pi}$,
$a \leftarrow a_0$, where $a_0 \sim \hat{\pi}$
**for** $\ell = 1, \ldots, L$ **do**
  $a \leftarrow a + \alpha_{\text{step}} \nabla_a Q_\theta(s, a)$
**end for**
**return** $a$

**QFQL [24]** also trains a behavior cloning actor via flow matching and critic using TD Learning, and perform policy optimization at test time with $Q$ critic gradeint guidance. QFQL uses the critic gradient at the intermediate noisy action $a_t$ directly (i.e., $\nabla_{a_t} Q(s, a_t)$), which is out-of-distribution for a critic trained only on clean actions. During inference, the denoising step at time $t$ is characterized by:

$$a_{t+\delta} \leftarrow a_t + \delta \left( v_\theta(s, a_t, t) + \frac{1}{\beta} \nabla_{a_t} Q(s, a_t) \right) \tag{19}$$

**RobustQ** is a novel baseline we consider, inspired by the adversarially robust classifiers [55, 26, 60, 22] in the text-to-image generation classifier guidance literature. To handle querying $Q$ gradient at the noisy action step $a_t$, we train a $Q$ function $Q(s, a_t, t)$ to regress on the IQL $Q$ with loss:

$$l_{robust} = (Q(s, a_t, t) - Q(s, a))^2. \tag{20}$$

Then, we query the gradient $\nabla_{a_t} Q(s, a_t, t)$ during inference time and use that as guidance.

**CFGRL [14]** also performs policy improvement during test time, but does not rely on critic gradient. During training, CFGRL trains a base flow matching policy $f(s, a_t, t)$ and an optimality-conditioned policy $f_{o=1}(s, a_t, t)$. $o$ is a binary indicator on whether the policy is "optimal", and $o = 1$ means it is. Following the original paper, we define "optimal" as $0 < A(s, a) = Q(s, a) - V(s)$. The base policy $f$ maps Gaussian noise to the whole dataset distribution, while the conditioned policy $f_{o=1}$ maps noise to the subset of the dataset distribution that is "optimal". During inference, CFGRL performs classifier free guidance (CFG) to combine the two policies into a single guided policy:

$$v_{\text{cfg}}(s, a_t, t) = f(s, a_t, t) + w(f_{o=1}(s, a_t, t) - f(s, a_t, t)),$$

where $w$ is the CFG weight that controls the balance between the two velocities.

**(2) Training-time methods**

**Q-Score Matching (QSM) [53]** trains a diffusion policy to follow the Boltzmann distribution of the Q-function, $\pi(\cdot \mid s) \propto e^{Q(s,\cdot)/\beta}$ by approximating the score of intermediate noised actions $a_t = \sqrt{\alpha_t} a + \sqrt{1 - \alpha_t}\varepsilon$ directly with the action gradient of the critic:

$$\mathcal{L}_{\text{QSM}}(\theta) = \mathbb{E}_{\varepsilon\sim\mathcal{N},\, t\sim\mathcal{U}\{0,\frac{1}{T},\ldots,1-\frac{1}{T}\}} \left[ \left\| -\frac{1}{\beta} \nabla_{a_t} Q(s, a_t) - f_\theta(s, a_t, t) \right\|_2^2 \right]. \tag{21}$$

Note that QSM essentially hill climbs with the OOD gradient direction (*i.e.*, $\nabla_{a_t} Q(s, a_t)$) during training. In practice, we find that QSM alone often does not perform well in offline RL settings. Therefore, we try to improve its performance by augmenting the QSM objective with a standard DDPM behavior regularization term $\mathcal{L}_{\text{DDPM}}(\theta) = \mathbb{E}_{\varepsilon\sim\mathcal{N},\, t\sim\mathcal{U}\{0,\frac{1}{T},\ldots,1-\frac{1}{T}\}} \left[ \|\varepsilon - f_\theta(s, a_t, i)\|_2^2 \right]$. We call this agent QSM-BC. QSM-BC has the overall actor loss $\mathcal{L}_{\text{QSM-BC}}(\theta) = \mathcal{L}_{\text{QSM}}(\theta) + \eta \mathcal{L}_{\text{DDPM}}(\theta)$.

**Diffusion Actor-Critic (DAC) [13]** is similar to QSM, but incorporates a behavior prior constraint: it aims to learn policy $\pi(\cdot|s) \propto \hat{\pi}(\cdot|s) e^{Q(s,\cdot)}$. Therefore, they assume that

$$\nabla_{a_t} \log \pi(a_t \mid s) \approx \nabla_{a_t} \log \hat{\pi}(a_t \mid s) + \frac{1}{\beta} \nabla_{a_t} Q(s, a_t).$$

The actor is trained by matching $f_\theta$ to a linear combination of the behavior cloning target $\varepsilon$ and the Q-gradient:

$$\mathcal{L}_{\text{DAC}}(\theta) = \mathbb{E}_{\varepsilon\sim\mathcal{N},\, t\sim\mathcal{U}\{0,\ldots,T-1\}} \left[ \left\| \varepsilon - \frac{1}{\beta} \nabla_{a_t} Q(s, a_t) - f_\theta(s, a_t, t) \right\|_2^2 \right]. \tag{22}$$

DAC can be interpreted as a training-time version of QFQL, which ascends on the OOD gradient direction with a behavior constraint.

**Flow Q-Learning (FQL) [50]** avoids the issue of backpropagating through the denoising chain by distilling a flow matching behavior cloning policy $f_\theta : \mathcal{S} \times \mathbb{R}^A \times [0, 1] \to \mathbb{R}^A$ into a one-step policy $f_\omega : \mathcal{S} \times \mathbb{R}^A \to \mathbb{R}^A$. $f_\theta$ is trained with a standard flow matching objective, and the one-step policy objective is:

$$\mathcal{L}_{\text{onestep}}(\omega) = \mathbb{E}_{z\sim\mathcal{N}} \left[ \alpha_{\text{FQL}} \|f_\omega(s, z) - \text{ODE}(f_\theta(s, \cdot, \cdot), z)\|_2^2 - Q(s, f_\omega(s, z)) \right], \tag{23}$$

where $\alpha_{\text{FQL}}$ is a coefficient balancing proximity (in 2-Wasserstein distance) to the behavior policy $\pi_\theta$ with $Q$-value maximization.

**Q-learning with Adjoint Matching (QAM) [33]** addresses the instability of BPTT by leveraging adjoint matching [10] to directly incorporate the critic's action gradient into the flow policy training without back-propagating through the denoising process. Specifically, QAM targets the optimal behavior-constrained policy $\pi(a \mid s) \propto \hat{\pi}(a \mid s) e^{Q(s,a)/\beta}$ by solving the following stochastic optimal control (SOC) objective via the adjoint matching loss:

$$\mathcal{L}_{\mathrm{AM}}(\theta) = \mathbb{E}_{s,\{a_t\}_t}\left[\int_0^1 \left\| \frac{2(f_\theta(s,a_t,t) - \hat{f}(s,a_t,t))}{\sigma_t} + \sigma_t \tilde{g}_t \right\|_2^2 dt\right], \tag{24}$$

where $\tilde{g}_t$ is the "lean" adjoint state propagated backwards through the base flow model $\hat{f}$ via

$$d\tilde{g}_t = -\nabla_{a_t}\left[2\hat{f}(s,a_t,t) - a_t/t\right]\tilde{g}_t\, dt, \quad \tilde{g}_1 = -\nabla_{a_1} Q(s,a_1)/\beta. \tag{25}$$

Unlike approximation-based methods (e.g., QSM, DAC), the adjoint state computation uses only $\hat{f}$ rather than $f_\theta$, avoiding backpropagating through the policy that is being optimized.

## J. Hyperparameter and Tuning Details

For all the main results in Fig. 5, Fig. 7, and Fig. 8, we run 10 different seeds. All other ablations experiment include 4 seeds or more.

Table 2 below denote the common hyperparameters and Table 3 denote the method-specific hyperparameters. For each method, including QGF and all baseline methods, we tune the method-specific hyperparameter per domain in the ranges specified in Table 5. Specifically, we tune hyperparameters per domain on task 2 and 4 only, and re-run all 10 seeds on task 1 through 5. We choose this hyperparameter tuning since task difficulty gradually increases from task 1 to task 5, and so tuning on task 2 and 4 represent the average performance on the domain.

| Parameter | Value |
|---|---|
| Batch size | 1024 |
| Discount factor ($\gamma$) | 0.999 |
| IQL Critic Expectile ($\tau$) | 0.9 |
| Action chunk horizon ($h$) | 5 |
| Flow steps | 10 |
| Offline training steps | $5 \times 10^5$ |
| Evaluation episodes | 10 |
| Critic Network Size | [1024, 1024, 1024, 1024] |
| Actor Network Size | [1024, 1024, 1024, 1024] |
| Learning rate | 3e-4 |
| Critic ensemble size | 2 |
| Critic aggregation method | min |

**Table 1: Common hyperparameters** used for singletask Offline RL experiments on OGBench 100M datasets.

| Parameter | Value |
|---|---|
| Batch size | 4096 |
| Discount factor ($\gamma$) | 0.999 |
| Flow steps | 10 |
| Offline training steps | $1 \times 10^6$ |
| Evaluation episodes | 50 per task |
| Critic Network Size | [1024, 1024, 1024, 1024] |
| Actor Network Size | [1024, 1024, 1024, 1024] |
| Learning rate | 3e-4 |
| Critic ensemble size | 2 |
| Critic aggregation method | mean for puzzle-*, min for cube-* |

**Table 2: Common hyperparameters** used for offline goal-conditioned RL experiments on OGBench.

| | *Training-time* | | | | | *Test-time* | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Domains | DAC ($\alpha$) | QSM +BC ($\tau_{\text{inv}}, \alpha$) | FQL ($\alpha$) | QAM ($\tau_{\text{inv}}$) | EDP ($w_{\text{BC}}$) | Grad Step ($\alpha_{\text{step}}, L$) | CFGRL ($\tau_{\text{cfg}}$) | BPTT ($\tau_{\text{g}}$) | QFQL ($\tau_{\text{g}}$) | Robust Q ($\tau_{\text{g}}$) | QGF Jac. ($\tau_{\text{g}}$) | **QGF** ($\tau_{\text{g}}$) | **QGF BFN** ($\tau_{\text{g}}$) |
| cube-triple-100M-* | 100 | (0.1, 10) | 1000 | 0.1 | 100 | (0.01, 3) | 3.0 | 0.06 | 0.06 | 0.12 | 0.04 | 0.06 | 0.02 |
| cube-quadruple-100M-* | 100 | (0.1, 10) | 1000 | 0.1 | 300 | (0.01, 3) | 3.0 | 0.06 | 0.06 | 0.10 | 0.02 | 0.10 | 0.008 |
| puzzle-4x4-100M-* | 100 | (0.1, 10) | 1000 | 0.1 | 300 | (0.01, 3) | 2.0 | 0.008 | 0.02 | 0.10 | 0.008 | 0.02 | 0.004 |
| scene-100M-* | 300 | (0.1, 10) | 300 | 0.1 | 30 | (0.01, 5) | 5.0 | 0.12 | 0.04 | 0.12 | 0.008 | 0.12 | 0.02 |

**Table 3: Domain-specific hyperparameters** for single-task OGBench experiments.

| | Guidance weight ($\tau_g$) | | | | DQC critic | | | |
|---|---|---|---|---|---|---|---|---|
| Domains | BPTT | QFQL | QGF-Jacobian | QGF | $h_c$ | $h_a$ | $\tau$ | $\kappa_d$ |
| puzzle-4x5-3M | 10 | 10 | 3 | 3 | 25 | 5 | 0.9 | 0.5 |
| cube-triple-100M | 3 | 3 | 3 | 3 | 25 | 5 | 0.93 | 0.8 |
| cube-quadruple-100M | 10 | 3 | 10 | 3 | 25 | 5 | 0.93 | 0.8 |
| puzzle-4x6-1B | 1 | 1 | 1 | 1 | 25 | 1 | 0.7 | 0.5 |
| cube-octuple-1B | 10 | 10 | 10 | 10 | 25 | 5 | 0.93 | 0.5 |

**Table 4: Domain-specific guidance weights and DQC value function hyperparameters** for goal-conditioned RL experiments on OGBench. $h_c$ and $h_a$ are the critic and policy action chunk horizons; $\kappa_b$ and $\kappa_d$ are the backup and distillation expectiles used by the Decoupled Q-Chunking (DQC) critic. These are tuned values taken from the DQC paper.

| Method | Hyperparameter(s) | Tuning Range |
|---|---|---|
| *Training-time methods* | | |
| DAC | $\alpha$ | $\{100, 300, 1000\}$ |
| QSM+BC | $(\tau_{\text{inv}}, \alpha)$ | $\{0.1, 3.0\} \times \{10, 30\}$ |
| FQL | $\alpha$ | $\{10, 100, 300, 1000\}$ |
| QAM (IQL) | $\tau_{\text{inv}}$ | $\{0.1, 0.3, 1.0\}$ |
| EDP | $w_{\text{BC}}$ | $\{1, 10, 30, 100, 300\}$ |
| CFGRL | $\tau_{\text{cfg}}$ | $\{1.0, 2.0, 3.0, 5.0, 10.0, 15.0\}$ |
| *Test-time methods* | | |
| GradStep | $(\eta, K)$ | $\{0.01, 0.03, 0.1\} \times \{1, 3, 5\}$ |
| QGF | $\tau_g$ | $\{0.004, 0.008, 0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.12\}$ |
| QFQL | $\tau_g$ | $\{0.004, 0.008, 0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.12\}$ |
| BPTT | $\tau_g$ | $\{0.004, 0.008, 0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.12\}$ |
| BFN | $N$ | $\{4, 8, 16, 32\}$ |
| QGF+BFN | $(N, \tau_g)$ | $\{4, 8, 16, 32\} \times \{0.004, 0.008, 0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.12\}$ |

**Table 5: Hyperparameter tuning ranges** for OGBench experiments.

## K. Limitations

Here we discuss the limitations of our proposed RL algorithm QGF. As discussed, our method relies on using the critic gradient to guide action optimization at test time. While this is generally much cheaper than other test-time RL algorithms such as best-of-N sampling, a critic parameterized by a large model may be expensive to take gradients over. Further work is needed in this direction to optimize the inference compute requirement of our method for large critics. Second, QGF relies on adjusting the output of a base reference model. If the base model is under-trained or does not well represent the data distribution, using QGF to improve at test time would not be efficient. One interesting direction of future research is better training recipe for reference models for QGF.