`floq`: Training Critics via Flow-Matching for Scaling Compute in Value-Based RL

Bhavya Agrawalla
Carnegie Mellon University
`bbagrawa@andrew.cmu.edu`

Michal Nauman
University of Warsaw
`Nauman.mic@gmail.com`

Khush Agrawal
Carnegie Mellon University
`khusha@andrew.cmu.edu`

Aviral Kumar
Carnegie Mellon University
`aviralku@andrew.cmu.edu`

Abstract

A hallmark of modern large-scale machine learning techniques is the use of training objectives that provide dense supervision to intermediate computations, such as teacher forcing the next token in language models or denoising step-by-step in diffusion models, enabling models to learn complex functions in a generalizable manner. Motivated by this observation, we investigate the benefits of iterative computation for temporal difference (TD) methods in reinforcement learning (RL), which typically represent value functions in a monolithic fashion. We introduce `floq` (*flow-matching Q-functions*), an approach that parameterizes the Q-function using a velocity field and trains it using techniques from flow-matching, typically used in generative modeling. This velocity field is trained using TD-learning, which bootstraps from values produced by a target flow, computed by running multiple steps of numerical integration. `floq` allows for more fine-grained control and scaling of the Q-function capacity than monolithic architectures, by appropriately setting the number of integration steps. Across a suite of 50 challenging offline RL and online fine-tuning tasks, `floq` demonstrates superior performance, improving by $\sim 2\times$ in hard tasks. `floq` scales capacity far better than standard Q-function architectures, highlighting the potential of iterative computation for value learning.

Code and runs for `floq` can be found at: https://github.com/CMU-AIRe/floq

1 Introduction

A key principle in building effective models in various areas of machine learning is the use of ***iterative computation***: producing complex output functions by composing a sequence of simpler operations. E.g., language models based on transformers [68] can generate coherent text by predicting the next token or by composing atomic reasoning strategies [19]. Similarly, diffusion and flow-matching models [2, 27, 44, 65] synthesize images by progressively denoising small perturbations. Effective results from these models suggests that iterative computation is a powerful tool for modeling complex functions with deep networks, by scaling compute appropriately.

Motivated by these results, in this paper, we ask: ***can iterative computation also improve value estimation in reinforcement learning (RL)?*** Specifically, we are interested in improving the estimation of the Q-value function. While Q-functions map state-action inputs to a scalar value, they are known to be highly complex and difficult to fit accurately (e.g., [12]). Standard temporal-difference (TD) learning used to train Q-functions struggles to leverage capacity of deep networks [6, 21, 36, 37, 47], often resulting in poor generalization. These problems are further exacerbated in the offline RL problem setting [34, 42], where we must learn entirely from static datasets. This motivates exploring architectures that spend compute iteratively to estimate value functions, potentially yielding better Q-values and policies.

A natural starting point for using iterative compute in value-based RL is to utilize a ResNet [25] Q-function, where stacking more residual blocks provides a way to run iterative computation. Recent work has obtained modest gains with ResNets [14, 38, 39, 50], but these methods need normalization and regularizers to enable stable training [6, 38, 40, 50]. Despite improvements, these approaches lack one ingredient that makes iterative computation effective in transformers or diffusion models: ***supervision at every step*** of the iterative process. Just as next-token prediction supervises each generated token and diffusion supervises each denoising step, we hypothesize that stepwise loss supervision applied to TD learning might lead to improvements.

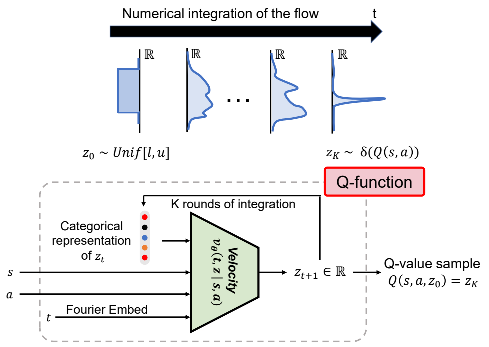
Numerical integration of the flow
t
ℝ
ℝ
ℝ
ℝ
$z_0 \sim Unif[l,u]$
$z_K \sim \delta(Q(s,a))$
Q-function
K rounds of integration
Categorical representation of z_t
Velocity $v_\theta(t, z \mid s, a)$
$z_{t+1} \in \mathbb{R}$
Q-value sample $Q(s,a,z_0) = z_K$
s
a
t
Fourier Embed

Figure 1: `floq` **architecture.** We model the Q-function via a velocity field of a flow-matching model. Over multiple calls, this velocity field converts a randomly sampled input $\boldsymbol{z}(0)$ into a sample from the Dirac-delta distribution centered at the mean Q-value. We build a flow-matching loss for training. Doing this enables us to scale computation by running numerical integration, with multiple calls to the velocity field. To train `floq`, we utilize a categorical representation of input $\boldsymbol{z}_t$ [14] and a Fourier representation of t.

With this observation, to effectively leverage iterative computation with dense supervision, we design a novel architecture for parameterizing Q-functions. Instead of using a single monolithic network, we represent the Q-function as a *velocity field* over a scalar value (Figure 1). Our approach, `floq` (flow-matching Q-functions) samples a scalar uniformly distributed noise and maps it to the Q-value by numerically integrating the predictions of the velocity field. We train the velocity with a linear flow-matching objective [2, 44], supervised to match the evolving TD-targets. At each step, we minimize the deviation between the current Q-value estimate and the corresponding TD-target. We introduce several design choices that stabilize training and help the architecture scale capacity effectively. These include appropriately setting the support of initial noise, using a categorical representation to handle non-stationary inputs and a Fourier time embedding to allow the velocity predictions to vary meaningfully across integration steps (Figure 2).

We use `floq` to represent the Q-function for a number of complex RL [34, 42] tasks from the OGBench [54] benchmark, previously studied by Park et al. [57]. In aggregate, we find that `floq` outperforms offline RL algorithms that represent Q-functions using a monolithic network by nearly $1.8\times$. `floq` is superior even when these approaches are provided with more parameters, and more complex and higher capacity architectures. `floq` also outperforms existing methods when running online fine-tuning after offline RL pre-training. We also show that increasing the number of flow-matching steps results in better downstream policy performance. Allocating the same capacity via Q-network ensembles or ResNets performs worse.

2 Related Work

Expressive generative models in RL. The most typical use of conventional generative models in RL has been to represent the policy, with several adoptions of diffusion policies [4, 24, 43, 61, 71, 77], flow-based policies [2, 44, 57], and sequence policies [30, 41, 76]. This shift is motivated by evidence that policy learning is often a significant bottleneck in offline RL [33, 53]. In parallel, policy-agnostic frameworks such as PA-RL [48] decouple algorithmic progress from specific architectural choices, enabling the use of diffusion, flows, or transformers interchangeably. Complementarily, we do not focus on policy expressivity and instead aim to utilize more expressive Q-functions, and opt to study `floq` on top of FQL for simplicity.

Scaling Q-functions. Efforts to scale Q-functions in RL have taken multiple directions with new training objectives such as classification losses [14, 38, 51, 64], architectures [7, 25, 39, 52], and regularization strategies [5, 36, 37, 46, 50],. Previous work has also attempted to develop scaling laws for TD learning [17, 62] and showing that alternatives to TD can scale to deeper architectures [70]. Despite these advances, a clear recipe for scaling value-based RL with TD-learning has yet to emerge.

Our work demonstrates that compute-efficient scaling can be realized not simply by increasing depth or width, but by introducing dense intermediate supervision through multiple integration steps of the Q-function. `floq` introduces a novel axis of scaling, allowing for compute scaling through additional integration steps rather than depth or width.

Scaling inference compute. A complementary line of work studies how more inference-time compute can be traded for performance. Classical MPC-style planners coupled with learned dynamics models such as PETS [8], MPPI [73], and PDDM [49] naturally allow scaling. In offline RL, MBOP [3] explicitly adopts planning with a learned model, a behavior prior, and a terminal value to extend the effective horizon. Generative world models enable similar test-time scaling by planning inside the learned model [30, 31]. Similarly, performance of MCTS-style methods improves with more simulation [10, 28, 63, 78]. Across all methods listed in this paragraph, the general pattern is that increasing test-time budget (i.e. simulations, horizon, candidate trajectories) improves returns up to the limit set by model bias and value estimation error. However, none of these works use more test-time compute to better estimate a value function. Our results show that `floq` can not only use more integration steps at inference time to amplify the "capacity" of the Q-function, but also that doing so during training helps us learn better Q-functions in the first place. We are the first to show that using more integration steps is a viable and effective path to scaling compute for critic networks.

3 Preliminaries and Notation

The goal in RL is to learn the optimal policy for an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \rho, \gamma)$. $\mathcal{S}, \mathcal{A}$ denote the state and action spaces. $P(s'|s,a)$ and $r(s,a)$ are the dynamics and reward functions. $\rho(s)$ denotes the initial state distribution. $\gamma \in (0,1)$ denotes the discount factor. Formally, the goal is to learn a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ that maximizes cumulative discounted value function, denoted by $V^\pi(s) = \frac{1}{1-\gamma}\sum_t \mathbb{E}_{a_t \sim \pi(s_t)}\left[\gamma^t r(s_t, a_t) | s_0 = s\right]$. The Q-function of a policy π is defined as $Q^\pi(s,a) = \frac{1}{1-\gamma}\sum_t \mathbb{E}_{a_t \sim \pi(s_t)}\left[\gamma^t r(s_t, a_t) | s_0 = s, a_0 = a\right]$, and we use Q^π_θ to denote the estimate of the Q-function of a policy π as obtained via a neural net with parameters θ. Value-based RL methods train a Q-network by minimizing the temporal difference (TD) error:

$$L(\theta) = \mathbb{E}_{(s,a,s') \sim \mathcal{D}, a' \sim \pi(\cdot|s')}\left[\left(r(s,a) + \gamma \bar{Q}(s',a') - Q_\theta(s,a)\right)^2\right], \tag{3.1}$$

where $\mathcal{D}$ is the offline dataset, $\bar{Q}$ is the target Q-network, s denotes a state, and a' is an action from policy $\pi(\cdot|s)$ that aims to maximize $Q_\theta(s,a)$. Offline RL methods are discussed in Appendix A.2.

4 `floq`: Training Q-Functions with Flow-Matching

In this section, we introduce the our proposed approach, `floq` (flow-matching Q-functions), which leverages iterative computation with dense supervision to train Q-functions. To do so, we address the two central questions needed to make `floq` work: **(a)** how to handle moving target values in the training loss for a flow-based Q-function and **(b)** how to do effective flow-matching over scalar Q-values without collapse for learning. Flow-matching preliminaries are discussed in Appendix A.2.

4.1 `floq` Parameterization

In contrast to standard deep Q-networks that map state-action pairs to scalar values, `floq` parameterizes a time-dependent, state-action-conditioned velocity field $v_\theta(t, \boldsymbol{z} \mid s, a)$ over a one-dimensional latent input $\boldsymbol{z} \in \mathbb{R}$. At $t = 0$, this input $\boldsymbol{z}$ is sampled from the uniform distribution $\mathrm{Unif}\,[l, u]$, where l and u are scalars that define the range of initial sample noise used for training. The velocity field transforms the initial sample $\boldsymbol{z}$ into a distribution over the Q-value. We will train `floq` such that the learned distribution of Q-values match a Dirac-Delta around the groundtruth Q-function, i.e., $\psi_\theta(1, \boldsymbol{z}|s,a) \sim \delta_{Q^\pi(s,a)}$ at $t = 1$. We can obtain the Q-value sample by numerically integrating the ODE using the Euler method. One instantiation is shown below. $\forall j \leq K$:

$$\psi_\theta(j/K, \boldsymbol{z} \mid s,a) = \boldsymbol{z} + \frac{1}{K}\sum_{i=1}^{j} v_\theta\left(\frac{i}{K}, \psi_\theta\left(i-1/K, \boldsymbol{z} \mid s,a\right) \Big| s,a\right),\ Q(s,a,\boldsymbol{z}) := \psi_\theta(1, \boldsymbol{z} \mid s,a) \tag{4.1}$$

An example illustration of this process is shown in Figure 1. This iterative process enables us to dynamically adjust the Q-function by varying the number of integration steps K, by controlling the number of evaluations of the velocity field v_θ, and thereby the "depth" of the model. Finally, we remark that although Equation 4.1 may appear similar to performing averaging like an ensemble, it is fundamentally different: the inputs passed to the velocity field v_θ at each step i depend on its own outputs from the previous step $i-1$. This recursive dependence introduces a form of iterative computation that is absent in conventional ensembles, that perform computation in parallel. As we demonstrate in our experiments (Section 5), this formulation enjoys greater benefits of scale than simply ensembling independent neural networks without iterative computation. In practice, the velocity field $v_\theta(^i/_K, \cdot|s,a)$ can be conditioned on the various representations of the intermediate Q-values $\psi_\theta(^{i-1}/_K, \cdot|s,a)$ to improve the effectiveness of learning. We opt to use a categorical representation of ψ_θ when passing it as input to the velocity network. We discuss this in Section 4.3.

4.2 Training Loss for the `floq` Architecture

With this parameterization in place, the next step is to design a training loss for the velocity field. Building upon TD-learning and flow-matching methods, a natural starting point is to iteratively train the velocity field using a loss that resembles linear flow-matching (Equation A.2), but with targets obtained via Bellman bootstrapping. This is akin to TD-flows [15] and γ-models [29] that train a generative dynamics model with TD-bootstrapped targets. To do so, we introduce a target velocity field $\tilde{v}_\theta(t, \boldsymbol{z} \mid s, a)$, parameterized as a stale moving average of the main velocity field v_θ, similar to target networks in standard value-based RL. Given a transition (s, a, r, s'), we first sample an action $a' \sim \pi(\cdot \mid s')$ from the current policy at the next state s', and compute target Q-value samples $\psi_{\tilde{\theta}}(1, \boldsymbol{z}' \mid s', a')$ by integrating the target flow, starting from some $\boldsymbol{z}'$ (via Euler integration) to obtain the predicted Q-value sample, $\psi_{\tilde{\theta}}(1, \boldsymbol{z}' \mid s', a')$.

We then average these predicted Q-value samples $\psi_{\tilde{\theta}}(1, \boldsymbol{z}' \mid s', a')$ for several values of the initial noise $\boldsymbol{z}'$ to compute an estimate of the target expected Q-value $Q_{\tilde{\theta}}(s', a')$. The bootstrapped TD-target is given by: $y(s,a) = r(s,a) + \gamma \frac{1}{m} \sum_{j=1}^{m} \psi_{\tilde{\theta}}(1, \boldsymbol{z}'_j \mid s', a')$, where $r(s,a)$ denotes the reward estimate for transition (note that this is distinct from distributional RL). We use this mean Q-target to train the Q-value at state-action pair (s,a) by regressing to the target $y(s,a)$ via a linear flow-matching loss. Concretely, given a $t \sim \text{Unif}[0,1]$, we construct an **interpolant** between noise $\boldsymbol{z}$ sampled at the initial step and the target Q-value y, $\boldsymbol{z}(t) = (1-t)\cdot\boldsymbol{z} + t \cdot y(s,a)$, and train the velocity at this interpolant to match the displacement from $\boldsymbol{z}(0)$ to y via flow-matching (Equation A.2):

$$\mathcal{L}_{\texttt{floq}}(\theta) = \mathbb{E}_{\boldsymbol{z},t}\left[\left\| v_\theta(t, \boldsymbol{z}(t) \mid s, a) - \frac{(y(s,a) - \boldsymbol{z})}{1-0} \right\|_2^2\right]. \tag{4.2}$$

4.3 Preventing Flow Collapse: How to Make `floq` Work Well?

So far, we have introduced a conceptual recipe for parameterizing and training a Q-function critic via flow matching. However, a naïve instantiation of this idea performed no better than a standard monolithic Q-function in our initial experiments. This performance is the result of the inability of the network to meaningfully condition on the interpolant $\boldsymbol{z}(t)$, leading the flow model to often collapse to a monolithic Q-network (Figure 2). Interestingly, we find that this problem can is a result of two peculiarities associated with applying flow-matching to TD: training with constantly evolving targets and running the flow on a scalar Q-values. We describe our approach for handling these pathologies, and to do so, we first answer: *what constitutes a "healthy"* `floq` *velocity field?* Then we introduce two crucial modifications to the `floq` architecture that enable learning healthy `floq` networks.

When is `floq` ***effective?*** Unlike traditional applications of flows, `floq` applies them to scalar Q-values. How does flow matching on a scalar work? Consider the trajectory traced by the flow during inference, as it evolves from initial noise ($t = 0$) to the Q-value estimate produced by the network ($t = 1$) (Figure 2; left). If this trajectory is a straight line, the velocity field $v_\theta(\boldsymbol{z}(t), t)$ does not need to depend on t and predicting a constant velocity proportional to the target Q-value is sufficient. In this case, flow matching provides no additional capacity beyond a monolithic Q-network. In contrast, if the trajectory is curved, the velocity field must utilize the interpolant $\boldsymbol{z}(t)$ and time t to predict customized velocities and be able to integrate to an accurate Q-value estimate at $t = 1$. Thus, even though training uses a simple linear flow-matching loss, extra capacity emerges only when the learned flows produce (slightly) curved trajectories. Here, iterative computation amplifies model capacity, allowing `floq` to outperform monolithic Q-networks. Note that overly curved flows are also problematic as they amplify errors in the integration process itself. Therefore, we want to attain an intermediate sweet spot in regards to the straightness of the traversals (see Figure 12).

Design choice 1: Distribution of the initial noise sample. As shown in prior works [44, 45], rescaling the source noise leaves the target distribution unchanged, but it alters the curvature of the transport trajectories. Interestingly, this effect seems to be particularly pronounced when applying

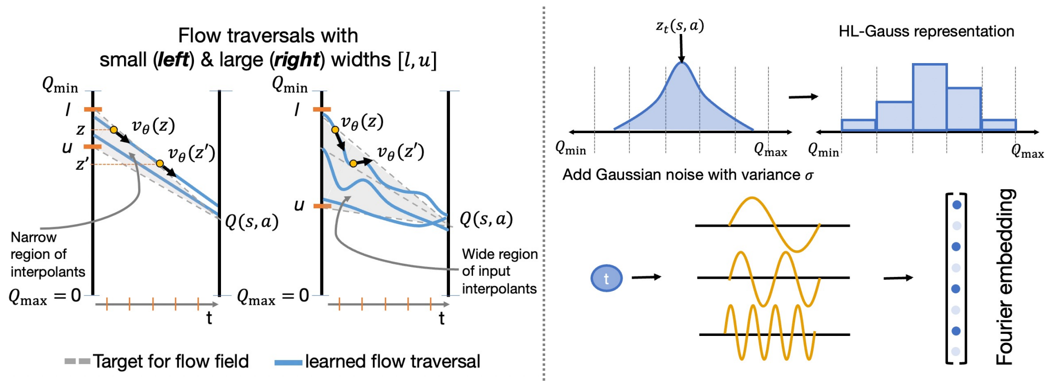

Figure 2: ***Illustrating the role of our design choices.*** **Left:** When the width of the interval $[l, u]$ is small, and the overlap between this interval and the range of target Q-values we hope to see is minimal, we would expect to see more straight flow traversals, that might be independent of interpolant $\boldsymbol{z}$. However, with wider intervals $[l, u]$, the flow traversal would depend on $\boldsymbol{z}$, and hence span a curved path when running numerical integration during inference. **Right:** Illustrating how we transform an input interpolant $\boldsymbol{z}$ into a categorical representation (top) and converting time t into a Fourier-basis embedding (bottom).

flow-matching to scalar TD-learning (see Figure 12 in experiments). As such, we find that that setting the bounds l and u for the distribution of the initial noise, $\text{Unif}\ [l, u]$ greatly affects the performance of `floq`.

We hypothesize that two aspects are important: (a) how close the target Q-values during training are to the chosen interval $[l, u]$, and (b) the width of the interval $u - l$. If the width $u - l$ is too small, then the interpolants $\boldsymbol{z}(t)$ span only a very limited range of values. When we then run (imperfect) TD-loss training on these interpolants, the network parameterizing the velocity field receives little meaningful variation in $\boldsymbol{z}(t)$ to associate changes in target values with. As a result, the model fails to exploit $\boldsymbol{z}(t)$ effectively and degenerates into behaving like a standard monolithic Q-function. Likewise, if the interval $[l, u]$ is very disjoint from the range of target Q-values during training, then all interpolants $\boldsymbol{z}(t)$ are forced to predict large velocities pointing in the general direction of the target Q-value. This reduces the need to learn calibrated velocity predictions conditioned on $\boldsymbol{z}(t)$ and time t. We show this in Figure 2, left.

Thus, we propose to choose l and u using a simple heuristic. We set $u = Q_{\max}$ (=0 for most of our tasks). We then choose $l \geq Q_{\min}$ to maximize the interval width $u - l$ while yielding stable learning curves and TD-error values comparable to a monolithic network. Here $Q_{\min}$ and $Q_{\max}$ denote the minimal and maximal possible Q-value achievable on the task. Thus, the interval $[l, u]$ is likely to overlap well with the target Q-values the velocity field must predict during training. We remark that this heuristic relies on the assumption that over the course of learning, Q-value predictions will evolve from near-zero values to the target value. Since standard network initializations already produce values near zero, this assumption is not limiting and helps cover the target Q-value range we see.

Design choice 2: Representing interpolant inputs to the velocity network. The second challenge arises because the magnitudes of the scalar interpolant $\boldsymbol{z}(t)$ evolve during training. While standard TD-learning naturally handles non-stationary *outputs* (Q-values growing from near-zero random initialization), this is usually manageable with best practices such as activation normalization [50]. In contrast, `floq` must cope with non-stationarity at the *input*, since the interpolant $\boldsymbol{z}(t)$ is fed into the velocity flow. As training progresses, its magnitude grows, leading to large gradients and activations in the network. To address this, we adopt a categorical representation of *input* $\boldsymbol{z}(t)$ (output is still scalar like baselines), inspired by the HL-Gauss encoding of Farebrother et al. [14], which prevents $\boldsymbol{z}(t)$'s magnitude from skewing activations. Concretely, we add Gaussian noise with standard deviation σ, then convert the resulting PDF $\mathcal{N}(\boldsymbol{z}(t), \sigma^2)$ into a categorical histogram over N bins spanning the expected Q-value range. In contrast to Farebrother et al. [14], we use a larger σ, such that roughly 80% of bins receive non-zero mass at initialization, encouraging broader coverage. Finally, we also utilized a Fourier basis representation of the time variable t, provided as input to the velocity network $v_\theta(t, \boldsymbol{z}(t))$. This is illustrated in Figure 2 (right). We show in our experiments than doing so helps substantially by encouraging the network to meaningfully utilize this time.

Summary: `floq` architecture and training

`floq` parameterizes the Q-function via a learned velocity field, trained with a linear flow-matching loss against the target Q-value (Eq.4.2). To scale capacity, we sample initial noise $\text{Unif}[l, u]$ broadly to overlap with target Q-values. The interpolant is encoded categorically, and the input t to v_θ is Fourier-encoded. See Algorithm1 for details.

5 Experimental Evaluation

The goal of our experiments is to evaluate the efficacy of `floq` in improving offline RL and online fine-tuning. To this end, we compare `floq` to state-of-the-art methods, and answer the following questions: **(1)** Does `floq` improve performance when compared to using similar-sized networks on benchmark tasks? and **(2)** How does the use of iterative computation via `floq` compare with the use of "parallel" computation of a neural network ensemble and "sequential" iterative computation driven by ResNets of comparable size? We then run several experiments to understand the behavior of `floq` critics. Furthermore, we run a variety of ablation studies to understand the design choices that drive the efficient use of iterative computation, including the roles of **a)** tuning the width of initial noise sample, **b)** categorical representations of the interpolant input, and **c)** Fourier-basis time embeddings.

5.1 Main Offline RL Results

Offline RL tasks and datasets. Following evaluation protocols from recent work in offline RL [13, 57, 69], we use the **OGBench** task suite [54] as our main evaluation benchmark (see Figure 16). OGBench provides a number of diverse, challenging tasks across robotic locomotion and manipulation, where these tasks are generally more challenging than standard D4RL tasks [16], which have been saturated as of 2024 [53, 60, 67]. While OGBench was originally designed for benchmarking offline goal-conditioned RL, we use its reward-based single-task variants ("`-singletask`" from Park et al. [57]). We employ 5 locomotion and 5 manipulation environments where each environment provides 5 tasks, totaling to **50** state-based OGBench tasks. Some tasks are more challenging and longer-horizon (e.g., marked in the table).

Comparisons and evaluation protocol. In addition to a `floq` critic, our experiments use flow-matching policies. Thus, flow-Q learning (**FQL**) [57], which utilizes a flow-matching policy with a monolithic Q-network, is our main comparison. FQL reports results with 1M training steps; we additionally re-run FQL for 2M steps and report both results. We run `floq` with a default set of hyperparameters across tasks, but on the more challenging `humanoidmaze-large` and `antmaze-giant` tasks we found a larger batch size of 512 to be more effective, which we use for both FQL and `floq`. Beyond FQL, we compare against three recent SOTA offline RL algorithms, all of which rely on monolithic Q-functions: (i) **ReBRAC** [67], the strongest-performing method with a monolithic Q-network and a Gaussian policy; (ii) **DSRL** [69] that adapts a diffusion-based behavior cloning policy by performing RL over its latent noise space, improving over FQL; and (iii) **SORL** [13] that leverages shortcut flow models to improve upon FQL. Note that none of them utilize a flow-matching Q-function, but do innovate across various properties of policy training. Comparing to the strongest methods that innovate on the policy allows us to evaluate the importance of a flow-matching Q-function.

`floq` **configuration.** We run `floq` in two configurations. The "default" configuration utilizes the same hyperparameters across tasks, whereas the "best" configuration uses environment-specific hyperparameters that still are fixed across all tasks in that environment, to get a sense of the upper bound with `floq`. Even the default configuration of `floq` substantially outperforms all prior methods. The default configuration utilizes $K = 8$ flow steps and sets the width of $u-l$ to be $\kappa \times (Q_{\max} - Q_{\min})$, where $\kappa = 0.1$. The best configuration tunes the number of flow steps $K \in \{4, 8\}$ and $\kappa \in \{0.1, 0.25\}$ per environment (*not* per task). For a fair comparison with FQL, we use a 4-layer flow critic in all environments, except in the cube (single and double) environments where we employ a smaller 2-layer flow critic because we saw training instabilities with 4-layer critics.

Empirical results. Observe in Table 1 that `floq` outperforms prior methods, including FQL, on average across all 50 tasks, evaluated over 3 seeds for each task. Also note that `floq` improves over FQL most on the harder environments, where FQL attains performance below 50% success rate (antmaze-giant, hmmaze-large, cube-double, puzzle-3x3, and puzzle-4x4). For a statistically rigorous evaluation, we adopt techniques from Agarwal et al. [1] and plot various statistics of the comparisons between `floq` and FQL: **1)** median and IQM scores in Figure 8, where we do not observe *any* overlap

Table 1: **Offline RL results (all tasks).** `floq` achieves competitive or superior performance compared to prior approaches. "Hard" environments refers to the set of environments where the FQL approach attains below 50% performance, averaged over the 5 tasks. `floq` is especially more performant on these hard environments over prior comparisons, where its performance (with best configuration) is around $1.8\times$ of FQL. We don't report DSRL [69] here as this prior work does not run on the exhaustive set of tasks (see Table 2 for these). A comparison on just the default tasks reveals `floq` outperforms DSRL by $> 2\times$.

	Gaussian Policy		Flow Policy			Flow Q-function (Ours)	
Env. (5 tasks each)	BC	ReBRAC	SORL	FQL (1M)	FQL(2M)	floq (Def.)	floq (Best)
antmaze-large	11 ±1	81 ±5	89 ±2	79 ±3	83 ±5	**91** ±5	**91** ±5
antmaze-giant *(Hard)*	0 ±0	26 ±8	9 ±6	22 ±19	27 ±23	36 ±21	**51** ±12
hmmaze-medium	2 ±1	22 ±8	64 ±4	57 ±5	69 ±20	**82** ±10	**82** ±10
hmmaze-large *(Hard)*	1 ±0	2 ±1	5 ±2	9 ±6	16 ±9	**28** ±9	**28** ±9
antsoccer-arena	1 ±0	0 ±0	**69** ±2	60 ±2	61 ±10	65 ±12	65 ±12
cube-single	5 ±1	91 ±2	97 ±1	96 ±1	94 ±5	**98** ±3	**98** ±3
cube-double *(Hard)*	2 ±1	12 ±1	25 ±3	29 ±2	25 ±6	**47** ±15	**47** ±15
scene	5 ±1	41 ±3	57 ±2	56 ±2	57 ±4	**58** ±6	**58** ±6
puzzle-3x3 *(Hard)*	2 ±0	21 ±1	– ±–	30 ±1	29 ±5	**37** ±7	**37** ±7
puzzle-4x4 *(Hard)*	0 ±0	14 ±1	– ±–	17 ±2	9 ±3	21 ±5	**28** ±6
Avg Score (All Envs.)	3	31	–	46	47	56	**59**
Avg Score (Hard Envs.)	1	15	–	21	21	34	**38**

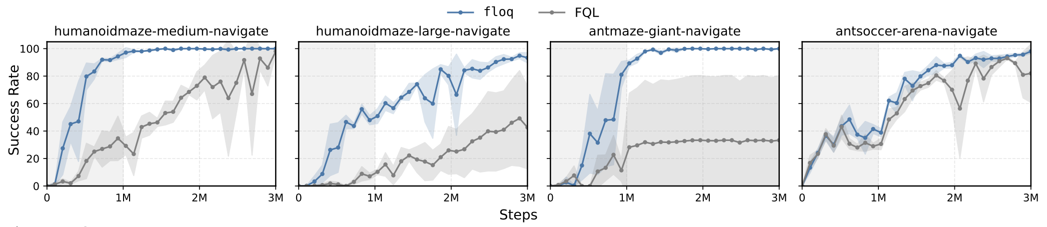

Figure 3: ***Learning curves for online fine-tuning*** of `floq` and FQL. `floq` not only provides a stronger initialization from offline RL training but also maintains its advantage throughout online fine-tuning on the hardest tasks, leading to faster adaptation, and higher final performance. The shaded area denotes offline RL.

between the confidence intervals; **2)** performance profile and $P(X > Y)$ statistic in Figure 9, which are both strictly in favor of `floq`. Since DSRL [69] only evaluates on the 10 default OGBench tasks, we also provide an additional results table on only the default tasks in each environment in the appendix (Table 2). On the default tasks, we find that `floq` outperforms DSRL (20% for DSRL vs. 45% for `floq`), improving by over $2\times$ in success rate. These results establish the efficacy of `floq`. While `floq` uses an expected Q-value backup it still learns a stochastic Q-function. Thus, we also compare `floq` to a representative distributional RL approach (IQN [9]) in Table 2, and we observe that `floq` outperforms IQN.

5.2 Main Online Fine-Tuning Results

Next, we evaluate `floq` in the online RL fine-tuning setting. Here, we first train agents completely offline for $1M$ steps and subsequently fine-tune them online for an additional $2M$ steps (Figure 3). Across four challenging tasks (humanoidmaze-medium, humanoidmaze-large, antmaze-giant, and antsoccer-arena), `floq` provides a substantially stronger initialization, faster learning during online interaction, and converges to higher final performance than FQL. Our complete set of results (Figure 10), show a similar trend establishing the efficacy of `floq` in online fine-tuning.

5.3 Understanding the Scaling Properties and Behavior of `floq`

To better understand the benefits of iterative compute in `floq`, we analyze its scaling behavior and compare it to different approaches. Specifically, we study: **(i)** how the number of flow-integration steps controls the expressivity of `floq`; **(ii)** how `floq`'s iterative computation compares to monolithic critic scaling; and **(iii)** the importance of applying supervision to the velocity field at every flow step.

1) How does the performance of* `floq` *depend on the number of integration steps for the flow? We now study the effect of varying the number of integration steps for the flow in `floq`. In Figure 4, we report the success rate of `floq` with $K \in \{1, 2, 4, 8, 16\}$ flow steps, alongside a monolithic Q-function (FQL) using the same architecture. Increasing the number of flow steps generally improves the performance of `floq`, with notable gains on harder tasks, `hmmaze-large` and `antmaze-giant`. Importantly, even with just 4 flow steps, `floq` already outperforms FQL, and the gap widens further

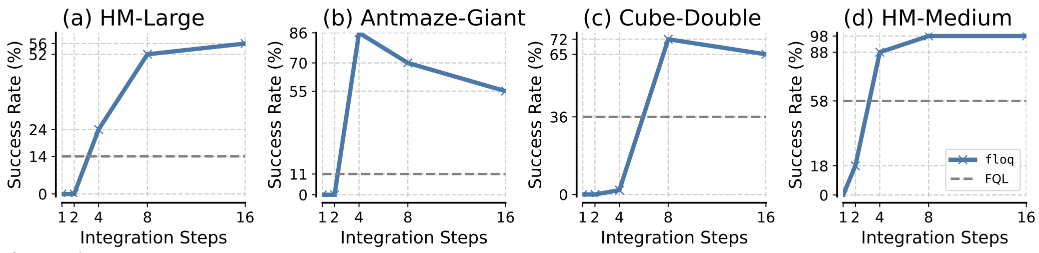

Figure 4: ***Effect of integration steps on*** `floq`. Performance of `floq` with varying flow steps, compared against a monolithic Q-function (FQL). More flow steps generally improve performance, but too many steps can lead to diminishing or negative returns (e.g., `antmaze-giant`). That said, in all configurations, `floq` outperforms FQL, and utilizing a moderately large number of flow steps is important.

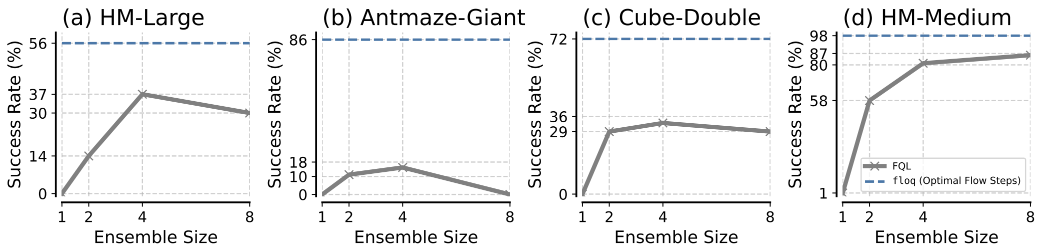

Figure 5: **Comparison of `floq` with monolithic ensembles.** We evaluate ensembles of size 1, 2, 4, and 8 of FQL critics. While larger ensembles do better, even an 8-critic ensemble falls short of `floq` with the same number of flow steps, showing that flow critics provide gains beyond parallel compute.

with additional steps. However, we also observe diminishing returns and, in some cases, slight degradation beyond a moderate number of steps (e.g., on `antmaze-giant`, where 8 and 16 steps perform worse than 4). We suspect that this degradation stems from overfitting when the number of integration steps for computing the target is excessive, and often manifests as unstable dynamics of TD-errors. A similar degradation is observed for ResNets in Figure 6.

3) How does `floq` compare against increasing "sequential" compute of monolithic critics? One hypothesis is that `floq` could be implementing a similar iterative computation strategy such as ResNet [25]. Unlike `floq`, ResNet does not utilize dense supervision after each computation block. To test whether this matters, we compare `floq` to a ResNet. Specifically, we use the same 4-layer flow critic from before and benchmark it against the best-performing ResNets with FQL. We ran a "cross-product" over possible ResNet configurations with various block sizes $(2, 4, 8, 16$ layers) and blocks $(8, 4, 2)$. These configurations represent all ways to build a ResNet where 32 layers are involved in a forward pass.

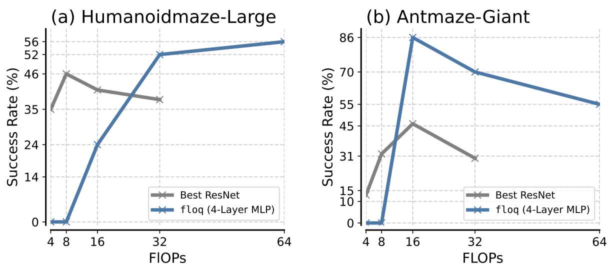

Figure 6: ***Comparison of `floq` with ResNet critics on the hardest tasks***: `hmmaze-large`, `antmaze-giant`. A 4-layer flow critic outperforms the best ResNets under a total budget on forward pass capacity, even after tuning over multiple residual configurations. For any given value on the x-axis, we plot the performance of the best performing ResNet configuration at that inference cost. For `floq`, we run more integration steps at inference. Thus, our approach of training `floq` does not simply add more depth.

We compare `floq` to the best ResNet configuration under a given total inference compute budget (i.e., given an upper bound on number of feed-forward layers) on humanoidmaze-large and antmaze-giant, since these hard environments should benefit from bigger ResNets. Observe in Figure 6, while ResNet critics do improve over FQL, they still remain worse than `floq`, even under matched inference compute. Note that `floq` does not itself utilize a ResNet (though its velocity network could use residual layers). Also note that while ResNet architectures instantiate a new set of parameters for every new layer added into the network, `floq` still only utilizes parameters from a *single* 4-layer MLP for all values of inference capacity. This indicates that the gains of flow critics cannot be attributed to adding more residual layers or more parameters, but rather the dense supervision provided by supervising the velocity field at each step of iterative computation. It also indicates that flow critics can perform better by scaling test-time compute without any extra parameters.

2) How does `floq` compare to scaling monolithic critics with ensembles? A common way to expand Q-function capacity is through ensembling—averaging predictions from multiple critics for backups and policy updates. This increases parallel rather than sequential compute, and is attractive if effective. We trained ensembles of 1, 2, 4, and 8 critics (each the same size as the base FQL critic) and compared them to `floq`. As shown in Figure 5, ensembles yield only modest gains over FQL, and even 8 critics fail to match `floq`. Notably, `floq` also uses 8 forward passes (via integration steps), so the compute is comparable. Thus, the benefits of flow critics stem not from parallel averaging.

4) Does `floq` benefit from densely supervising the velocity at all time steps? To answer this question, we trained a variant of `floq` where we supervised the velocity field was supervised only at $t = 0$ and used only a single flow step. Note that while one might think that training `floq` at $t = 0$ is equivalent to baseline FQL, this is not the case. This is because while we do use only oneintegration step, the input to the velocity network is still a scalar noise. The velocity field is trained to predict the difference between the target Q-value and this input noise, *for all different values of this noise*. This presents several "auxiliary tasks" for fitting the Q-function as opposed to just one in baseline FQL (i.e., the noise is set to 0 only). We hypothesize training the network to fit these auxiliary tasks does result in representational benefits, consistent with conventional wisdom in TD-learning that relates auxiliary losses with representational benefits [46].

As shown in Figure 7, this restricted variant substantially outperforms FQL but consistently underperforms `floq`, which supervises velocity at all $t \in [0, 1]$ and leverages multiple steps. On the humanoidmaze-large environment, performance increases from 14% (FQL) to 49% with only $t = 0$ training of `floq`, but full `floq` achieves 56%. On antmaze-giant, the gap is more pronounced, with scores of 11%, 39%, and 86%, respectively. We observed similar patterns on hm-medium (58%, 94%, 98%) and cube-double (36%, 38%, 72%). We hypothesize that the gain from multiple steps on hm-medium is smaller because it is a simpler task. Thus, while $t = 0$ already brings notable benefits, supervising at all t and multi-step integration is important for unlocking the full potential of `floq`.

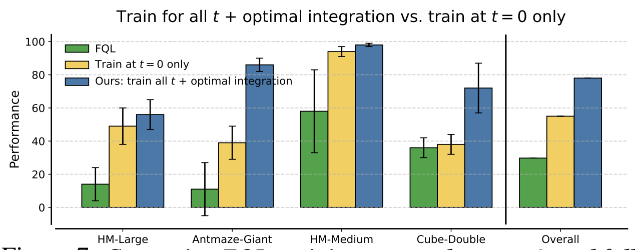

Figure 7: ***Comparing FQL, training `floq` only at*** $t = 0$***, and full*** `floq` with supervision across all t (and optimal integration steps k chosen from $\{4, 8, 16\}$). While $t = 0$ training improves over FQL, full `floq` consistently achieves the best performance, showing the benefits from training at all integration steps.

Takeaways: properties and behavior of `floq`

More integration steps are better, but performance saturates and can degrade at very high values. `floq` outperforms approaches for scaling parallel compute or sequential compute for monolithic Q-functions. `floq` outperforms monolithic Q-functions even with just one integration step, though multiple integration steps are required for best performance.

5.4 Ablation Studies for `floq`

Finally, in Appendix A.4, we present experiments ablating various design choices and hyperparameters in `floq`. Our goal is to evaluate the sensitivity of `floq` to these choices and prescribe thumb rules for tuning them. Concretely, the design choices we ablate in this section include: **a)** the range $[l, u]$ that provides the support for the initial noise sample $\boldsymbol{z}(0)$, **b)** the approach for embedding "time" of the flow step t and **c)** the approach for embedding the interpolant $\boldsymbol{z}(t)$. We present our results for **a)** here and for **b)**, **c)** in Appendix A.4, but the main findings for all are summarized below.

Takeaways: ablation studies for `floq`

1) Utilizing an HL-Gauss embedding for $\boldsymbol{z}(t)$ is crucial. Generally, the larger the coverage over bins, the better the performance of `floq` (Figure 13). 2) Utilizing a Fourier-basis embedding of time is critical for meaningfully conditioning on it (Figure 11). 3) A moderate width of the initial noise distribution improves flow curvature, and performs best (Figure 12).

Discussion, conclusion, and future work. This work introduced `floq`, a flow-matching approach to training critics that scales Q-function capacity through iterative integration and dense supervision, achieving state-of-the-art offline RL results and online fine-tuning results. Future directions include understanding how to set integration steps, exploiting `floq`'s cascaded family of critics for efficiency, and combining sequential test-time scaling with ensembles. Theoretically, it is important to study how curved flows enable error correction when learning Q-values for TD-learning.

Acknowledgements

We thank Max Sobol Mark, Zheyuan Hu, Aravind Venugopal, Seohong Park, Mitsuhiko Nakamoto, and Amrith Setlur for feedback on an earlier version of the paper and informative discussions. We thank members of the CMU AIRe lab for feedback and support. AK thanks Max Simchowitz for discussions on flow-matching and diffusion models. This work was supported by a Schmidt Sciences AI2050 fellowship and the Office of Naval Research under N00014-24-12206. We thank the TRC program of Google Cloud and NCSA Delta for providing computational resources that supported this work. This work used the Delta advanced computing and data resource at the National Center for Supercomputing Applications (NCSA) through allocation CIS250548 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services And Support (ACCESS) program. Delta is supported by the National Science Foundation (award OAC-2005572) and the State of Illinois, and ACCESS is supported by U.S. National Science Foundation grants 2138259, 2138286, 2138307, 2137603, and 2138296.

References

[1] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Neural Information Processing Systems (NeurIPS)*, 2021.

[2] Michael S Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. In *International Conference on Learning Representations (ICLR)*, 2023.

[3] Arthur Argenson and Gabriel Dulac-Arnold. Model-based offline planning. *arXiv preprint arXiv:2008.05556*, 2020.

[4] Arpit Bansal, Eitan Borgnia, Hong-Min Chu, Jie Li, Hamid Kazemi, Furong Huang, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Cold diffusion: Inverting arbitrary image transforms without noise. *Advances in Neural Information Processing Systems*, 36:41259–41282, 2023.

[5] Aditya Bhatt, Daniel Palenicek, Boris Belousov, Max Argus, Artemij Amiranashvili, Thomas Brox, and Jan Peters. CrossQ: Batch normalization in deep reinforcement learning for greater sample efficiency and simplicity. In *International conference on learning representations (ICLR)*, 2024.

[6] Johan Bjorck, Carla P Gomes, and Kilian Q Weinberger. Towards deeper deep reinforcement learning. *arXiv preprint arXiv:2106.01151*, 2021.

[7] Yevgen Chebotar, Quan Ho Vuong, Alex Irpan, Karol Hausman, F. Xia, Yao Lu, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, Keerthana Gopalakrishnan, Julian Ibarz, Ofir Nachum, Sumedh Anand Sontakke, Grecia Salazar, Huong Tran, Jodilyn Peralta, Clayton Tan, Deeksha Manjunath, Jaspiar Singht, Brianna Zitkovich, Tomas Jackson, Kanishka Rao, Chelsea Finn, and Sergey Levine. Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions. In *Conference on Robot Learning*, 2023.

[8] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

[9] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. *arXiv preprint arXiv:1710.10044*, 2017.

[10] Ivo Danihelka, Arthur Guez, Julian Schrittwieser, and David Silver. Policy improvement by planning with gumbel. In *International Conference on Learning Representations*, 2022.

[11] Sudeep Dasari, Oier Mees, Sebastian Zhao, Mohan Kumar Srirama, and Sergey Levine. The ingredients for robotic diffusion transformers. *arXiv preprint arXiv:2410.10088*, 2024.

[12] Kefan Dong, Yuping Luo, Tianhe Yu, Chelsea Finn, and Tengyu Ma. On the expressivity of neural networks for deep reinforcement learning. In *International Conference on Machine Learning*, pages 2627–2637. PMLR, 2020.

[13] 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.

[14] Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, et al. Stop regressing: Training value functions via classification for scalable deep rl. *arXiv preprint arXiv:2403.03950*, 2024.

[15] Jesse Farebrother, Matteo Pirotta, Andrea Tirinzoni, Rémi Munos, Alessandro Lazaric, and Ahmed Touati. Temporal difference flows. *arXiv preprint arXiv:2503.09817*, 2025.

[16] Justin Fu, Aviral Kumar, Ofir Nachum, G. Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *ArXiv*, abs/2004.07219, 2020.

[17] Preston Fu, Oleh Rybkin, Zhiyuan Zhou, Michal Nauman, Pieter Abbeel, Sergey Levine, and Aviral Kumar. Compute-optimal scaling for value-based deep rl. *arXiv preprint arXiv:2508.14881*, 2025.

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

[19] Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D. Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars, 2025. URL https://arxiv.org/abs/2503.01307.

[20] Divyansh Garg, Joey Hejna, Matthieu Geist, and Stefano Ermon. Extreme q-learning: Maxent rl without entropy. In *International Conference on Learning Representations (ICLR)*, 2023.

[21] Caglar Gulcehre, Srivatsan Srinivasan, Jakub Sygnowski, Georg Ostrovski, Mehrdad Farajtabar, Matt Hoffman, Razvan Pascanu, and Arnaud Doucet. An empirical study of implicit regularization in deep offline rl. *arXiv preprint arXiv:2207.02099*, 2022.

[22] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *ArXiv*, abs/2501.12948, 2025.

[23] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. Technical report, 2018.

[24] 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.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[26] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *ArXiv*, abs/1606.08415, 2016.

[27] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Neural Information Processing Systems (NeurIPS)*, 2020.

[28] Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Mohammadamin Barekatain, Simon Schmitt, and David Silver. Learning and planning in complex action spaces. In *International Conference on Machine Learning*, pages 4476–4486. PMLR, 2021.

[29] Michael Janner, Igor Mordatch, and Sergey Levine. γ-models: Generative temporal difference learning for infinite-horizon prediction. In *Advances in Neural Information Processing Systems*, 2020.

[30] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2021.

[31] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, 2022.

[32] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[33] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*.

[34] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11761–11771, 2019.

[35] 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.

[36] Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *International Conference on Learning Representations*, 2021.

[37] Aviral Kumar, Rishabh Agarwal, Tengyu Ma, Aaron Courville, George Tucker, and Sergey Levine. DR3: Value-based deep reinforcement learning requires explicit regularization. *International Conference on Learning Representations*, 2022.

[38] Aviral Kumar, Rishabh Agarwal, Xinyang Geng, George Tucker, and Sergey Levine. Offline Q-learning on diverse multi-task data both scales and generalizes. In *International Conference on Learning Representations*, 2023.

[39] Aviral Kumar, Anikait Singh, Frederik Ebert, Yanlai Yang, Chelsea Finn, and Sergey Levine. Pre-training for robots: Offline rl enables learning new tasks from a handful of trials. *RSS 2023; arXiv:2210.05178*, 2023.

[40] Hojoon Lee, Dongyoon Hwang, Donghu Kim, Hyunseung Kim, Jun Jet Tai, Kaushik Subramanian, Peter R Wurman, Jaegul Choo, Peter Stone, and Takuma Seno. SimBa: Simplicity bias for scaling up parameters in deep reinforcement learning. *arXiv preprint*, 2024.

[41] Kuang-Huei Lee, Ofir Nachum, Mengjiao Yang, Lisa Lee, Daniel Freeman, Winnie Xu, Sergio Guadarrama, Ian Fischer, Eric Jang, Henryk Michalewski, et al. Multi-game decision transformers. *arXiv preprint arXiv:2205.15241*, 2022.

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

[43] Zechu Li, Rickmer Krohn, Tao Chen, Anurag Ajay, Pulkit Agrawal, and Georgia Chalvatzaki. Learning multimodal behaviors from scratch with diffusion policy gradient. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=vU1SiBb57j.

[44] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

[45] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

[46] Clare Lyle, Mark Rowland, Georg Ostrovski, and Will Dabney. On the effect of auxiliary tasks on representation dynamics. In *International Conference on Artificial Intelligence and Statistics*, pages 1–9. PMLR, 2021.

[47] Clare Lyle, Mark Rowland, Will Dabney, Marta Kwiatkowska, and Yarin Gal. Learning dynamics and generalization in deep reinforcement learning. In *International Conference on Machine Learning*, pages 14560–14581. PMLR, 2022.

[48] 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*, abs/2412.06685, 2024.

[49] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

[50] Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. Bigger, regularized, optimistic: Scaling for compute and sample-efficient continuous control. *Advances in Neural Information Processing Systems*, 2024.

[51] Michal Nauman, Marek Cygan, Carmelo Sferrazza, Aviral Kumar, and Pieter Abbeel. Bigger, regularized, categorical: High-capacity value functions are efficient multi-task learners. *arXiv preprint arXiv:2505.23150*, 2025.

[52] Johan Obando-Ceron, Ghada Sokar, Timon Willi, Clare Lyle, Jesse Farebrother, Jakob Foerster, Gintare Karolina Dziugaite, Doina Precup, and Pablo Samuel Castro. Mixtures of experts unlock parameter scaling for deep rl. *arXiv preprint arXiv:2402.08609*, 2024.

[53] Seohong Park, Kevin Frans, Sergey Levine, and Aviral Kumar. Is value learning really the main bottleneck in offline rl? *arXiv preprint arXiv:2406.09329*, 2024.

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

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

[56] Seohong Park, Qiyang Li, and Sergey Levine. Flow q-learning. *ArXiv*, abs/2502.02538, 2025.

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

[58] 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.

[59] Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *International Conference on Machine Learning (ICML)*, 2007.

[60] Rafael Rafailov, Kyle Beltran Hatch, Anikait Singh, Aviral Kumar, Laura Smith, Ilya Kostrikov, Philippe Hansen-Estruch, Victor Kolev, Philip J Ball, Jiajun Wu, et al. D5rl: Diverse datasets for data-driven deep reinforcement learning. In *Reinforcement Learning Conference (RLC)*, 2024.

[61] 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.

[62] Oleh Rybkin, Michal Nauman, Preston Fu, Charlie Snell, Pieter Abbeel, Sergey Levine, and Aviral Kumar. Value-based deep rl scales predictably. *arXiv preprint arXiv:2502.04327*, 2025.

[63] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[64] Younggyo Seo, Carmelo Sferrazza, Haoran Geng, Michal Nauman, Zhao-Heng Yin, and Pieter Abbeel. Fasttd3: Simple, fast, and capable reinforcement learning for humanoid control. *arXiv preprint arXiv:2505.22642*, 2025.

[65] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning (ICML)*, 2015.

[66] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations (ICLR)*, 2021.

[67] Denis Tarasov, Vladislav Kurenkov, Alexander Nikulin, and Sergey Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2023.

[68] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[69] Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. *arXiv preprint arXiv:2506.15799*, 2025.

[70] Kevin Wang, Ishaan Javali, MichaĹ Bortkiewicz, Benjamin Eysenbach, et al. 1000 layer networks for self-supervised rl: Scaling depth can enable new goal-reaching capabilities. *arXiv preprint arXiv:2503.14858*, 2025.

[71] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning.

[72] Ziyun Wang, Alexander Novikov, Konrad Zolna, Jost Tobias Springenberg, Scott E. Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Manfred Otto Heess, and Nando de Freitas. Critic regularized regression. In *Neural Information Processing Systems (NeurIPS)*, 2020.

[73] Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40 (2):344–357, 2017.

[74] Yifan Wu, G. Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *ArXiv*, abs/1911.11361, 2019.

[75] Haoran Xu, Li Jiang, Jianxiong Li, Zhuoran Yang, Zhaoran Wang, Victor Chan, and Xianyuan Zhan. Offline rl with no ood actions: In-sample learning via implicit value regularization. In *International Conference on Learning Representations (ICLR)*, 2023.

[76] Taku Yamagata, Ahmed Khalil, and Raul Santos-Rodriguez. Q-learning decision transformer: Leveraging dynamic programming for conditional sequence modelling in offline rl. In *International Conference on Machine Learning*, pages 38989–39007. PMLR, 2023.

[77] 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.

[78] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *Advances in neural information processing systems*, 34:25476–25488, 2021.

A Appendices

A.1 Discussion and Perspectives on Future Work

In this paper, we presented `floq`, an approach for training critics in RL using flow-matching. `floq` formulates value learning as transforming noise into the value function via integration of a learned velocity field. This formulation enables scaling Q-function capacity by utilizing more compute during the process of integration to compute the Q-function. As a result of utilizing a flow-matching objective for training, `floq` utilizes dense supervision at every step of the integration process. We describe some important design choices to train flow-matching critics to make meaningful use of integration steps. Through our experiments, we show that `floq` attains state-of-the-art results on a suite of commonly-used offline RL tasks, and outperforms other ways of expanding capacity of a Q-function (e.g., via a ResNet or monolithic Q-function ensemble). We also show the necessity of learning curved flow traversals to make effective use of capacity and utilizing the design choices we prescribe in this work.

Future work. We believe `floq` presents an exciting approach to scale Q-function capacity. Thus, there are a number of both theoretical and empirical open questions. From an empirical standpoint, it is important to understand how to appropriately set the number of integration steps as excessive steps may degrade Q-function quality. This degradation, however, is not localized to just flows but also to ResNets (Figure 6), indicating that this is perhaps a bigger issue with TD-learning. Another interesting direction is to build new methods and workflows for using Q-functions that rely on the property that `floq` inherently represents a "cascaded" family of critics with different capacities—all within one network. Can this property be used for tuning network size upon deployment, cross-validation of model size, or improving efficiency of policy extraction? Answering this question would be interesting for future work. Finally, `floq` also provides one possibility for sequential or "depth"-based test-time scaling for value functions. Studying how this sort of sequential scaling can be combined with parallel scaling (i.e., ensembles) and horizon reduction techniques [55] would be interesting as well.

From a theoretical standpoint, quantifying iterative computation properties of `floq` would be impactful: in principle, curved flow traversals should enable the critic network to spend more test-time compute (i.e., integration steps) to perform equivalents of "error correction" and "backtracking" from large language models (LLMs) [22], but now in the space of scalar, continuous values to better approximate the target Q-function. We believe formalizing this aspect would not only be impactful for value-based RL, but could also shed light on methods to use test-time compute in flow/diffusion models in other domains. Second, our results show that there are substantial representation learning benefits of `floq`. We believe that studying the mechanisms and differences between feature learning induced by `floq` compared to standard TD-learning with regression [37] or classification [14] would be interesting for future value-based methods. `floq` also provides a rich family of auxiliary tasks to train a critic, which provides another angle to explain and study its properties. All of these are impactful directions to study in future work.

A.2 Background and Preliminaries

We operate in the offline RL [42] problem setting, where the replay buffer $\mathcal{P}$ corresponds to a static dataset of transitions $\mathcal{D} = \{(s, a, r, s')\}$ collected using a behavior policy π_β. Our goal in this setting is to train a good policy using the offline dataset $\mathcal{D}$ alone. The Q-network, Q_θ is typically parameterized by a deep network (e.g., an MLP).

Offline RL algorithms. Offline RL methods aim to learn a policy that maximizes reward while penalizing deviation from the behavior policy π_β, in order to mitigate the challenge of distributional shift. This objective has been instantiated in various ways, including behavioral regularization [18, 34, 57, 67, 74], pessimistic value function regularization [35], implicit policy constraints [48, 58, 59, 72], and in-sample maximization [20, 33, 75]. While our proposed `floq` architecture for Q-function parameterization is agnostic to the choice of offline RL algorithm, we instantiate it on top of FQL [57] that utilizes a flow-matching policy to better model multimodal action distributions. FQL trains the Q-function using the standard temporal-difference (TD) error from soft actor-critic (SAC) [23], shown in Equation 3.1, and optimizes the policy to stay close to a behavior policy estimated via flow-matching.

Flow-matching. Flow-matching [2, 44, 45] is an approach for training generative models by integrating deterministic ordinary differential equations (ODEs), which also makes use of iterative computation. Flow-matching is often posed as a deterministic alternative to denoising diffusion models [27, 66] that utilize stochastic differential equations (SDEs) for generating complex outputs. Concretely, given a target data distribution $p(\boldsymbol{x})$ over $\boldsymbol{x} \in \mathbb{R}^d$, flow-matching attempts to fit a time-dependent ***velocity field***, $v_\theta(t, \boldsymbol{x}) : [0, 1] \times \mathbb{R}^d \to \mathbb{R}^d$ such that the solution $\psi_\theta(t, \boldsymbol{x})$ to the ODE:

$$\frac{d}{dt}\psi_\theta(t, \boldsymbol{x}) = v_\theta(t, \psi_\theta(t, \boldsymbol{x})), \quad \psi_\theta(0, \boldsymbol{x}(0)) = \boldsymbol{x}(0) \tag{A.1}$$

transforms samples $\boldsymbol{x}(0)$ from a simple base distribution (e.g., standard Gaussian or uniform, as we consider in this work) into samples from $p(\boldsymbol{x})$ at time $t = 1$. While there are several methods to train a velocity flow, perhaps the simplest and most widely-utilized approach is linear flow matching [44], which trains the velocity flow to predict the gradient obtained along the linear interpolating path between $\boldsymbol{x}(0)$ and $\boldsymbol{x}(1)$ at all intermediate points. Concretely, define $\boldsymbol{x}(0) \sim p_0(\boldsymbol{x})$ be a sample from a simple initial distribution, $\boldsymbol{x}(1) \sim p(\boldsymbol{x})$ be a sample from the target distribution, and $t \sim \mathrm{Unif}([0, 1])$, we define interpolated points as $\boldsymbol{x}(t) = (1 - t) \cdot \boldsymbol{x}(0) + t \cdot \boldsymbol{x}(1)$, and train the velocity field to minimize the squared error from the slope of the straight line connecting $\boldsymbol{x}(0)$ and $\boldsymbol{x}(1)$ as follows:

$$\min_\theta \; \mathbb{E}_{\boldsymbol{x}(0),\boldsymbol{x}(1),t}\left[\left\| v_\theta(t, \boldsymbol{x}(t)) - \frac{(\boldsymbol{x}(1) - \boldsymbol{x}(0))}{1 - 0}\right\|_2^2\right]. \tag{A.2}$$

After training the velocity field $v_\theta(t, \boldsymbol{x}(t))$, flow-matching runs numerical integration to compute $\psi_\theta(t, \boldsymbol{x}(0))$. This numerical integration procedure makes several calls to compute the velocity field. Each subsequent call runs the velocity field, $v_\theta(t, \boldsymbol{x}(t))$ on input $\boldsymbol{x}(t)$ generated as output from the previous call, representing a form of an iterative computation process.

A.3 Additional Results for `floq`

In this section, we provide some additional and complete results supplementing the ones in main paper.

1. Figure 8 presents median and IQM scores and Figure 9 presents performance profiles and $P(X > Y)$ statistic comparing `floq` with FQL on all the 50 tasks studied in the paper.
2. Figure 10 presents results for online fine-tuning on all 10 default tasks.

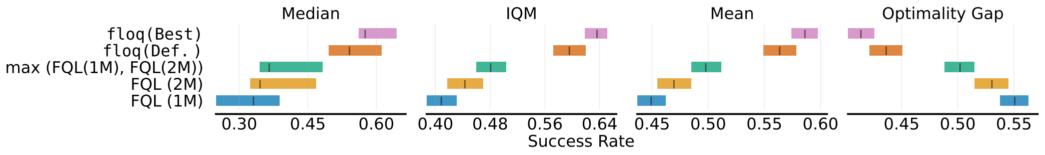

Figure 8: Comparison of `floq` against the baseline FQL across median, interquartile mean (IQM), mean and optimality gap, following Agarwal et al. [1]. Results show that `floq` consistently outperform FQL across all evaluation criteria with no confidence interval overlap in all cases, meaning that the gains from `floq` are significant.

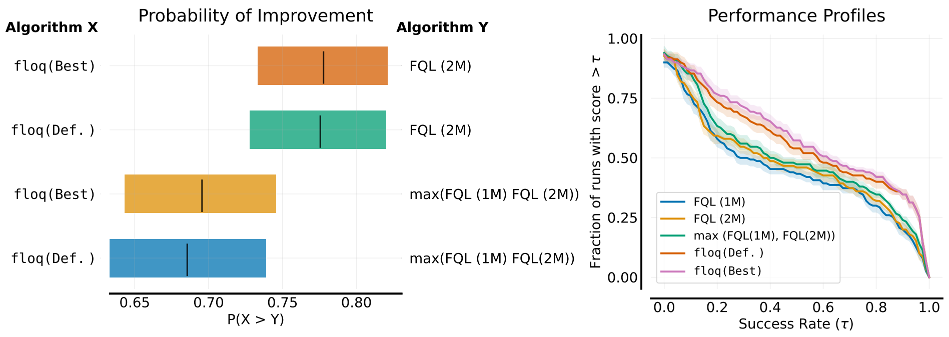

Figure 9: Comparison of `floq` against baseline FQL, following [1]. **Left:** Probability of Improvement $P(X > Y)$ showing that `floq` consistently outperform FQL across OGBench tasks. **Right:** Performance profiles illustrating that `floq` achieves higher scores across a larger fraction of runs compared to FQL.

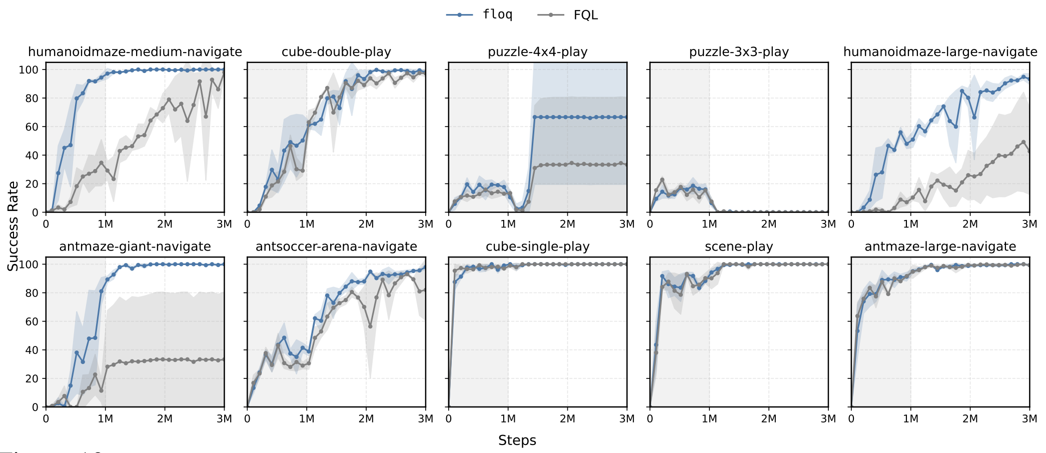

Figure 10: ***Learning curves for online fine-tuning*** of `floq` and FQL across all default tasks. `floq` not only provides a stronger initialization from offline RL training but also maintains its advantage through online fine-tuning, leading to faster adaptation and higher final success rates. The shaded gray area denotes offline RL training.

Table 2: **Offline RL results (Default Tasks).** `floq` achieves competitive or superior performance compared to the baselines. "Hard" tasks refers to the set of default tasks where the FQL baseline score is below 50% performance. `floq` is especially more performant on these hard tasks, more than doubling FQL's baseline performance.

	Gaussian Policy		Diff. Policy	Flow Policy				Flow Q-function (Ours)	
Env (Default Task)	BC	ReBRAC	DSRL	SORL	IQN	FQL (1M)	FQL(2M)	floq (Def.)	floq (Best)
antmaze-large	0 ±0	91 ±10	40 ±29	93 ±2	86 ±1	80 ±8	85 ±4	**94** ±4	**94** ±4
antmaze-giant	0 ±0	27 ±22	0 ±0	12 ±6	5 ±6	11 ±16	14 ±29	70 ±8	**86** ±4
hmmaze-medium	1 ±0	16 ±9	34 ±20	67 ±4	48 ±17	19 ±12	58 ±25	**98** ±1	**98** ±1
hmmaze-large	0 ±0	2 ±1	10 ±12	20 ±9	35 ±5	8 ±5	14 ±10	**52** ±8	**52** ±8
antsoccer-arena	1 ±0	0 ±0	28 ±0	**54** ±5	**54** ±7	39 ±6	49 ±11	49 ±10	49 ±10
cube-single	3 ±1	92 ±4	93 ±14	99 ±0	98 ±1	96 ±1	94 ±5	**99** ±2	**99** ±2
cube-double	0 ±0	7 ±3	53 ±14	33 ±8	57 ±2	36 ±6	29 ±8	**72** ±15	**72** ±15
scene	1 ±1	50 ±13	88 ±9	**89** ±9	80 ±4	76 ±9	78 ±7	83 ±10	83 ±10
puzzle-3x3	1 ±1	2 ±1	0 ±0	–	**20** ±3	16 ±5	14 ±4	17 ±6	17 ±6
puzzle-4x4	0 ±0	10 ±3	**37** ±13	–	16 ±1	11 ±3	5 ±2	12 ±4	19 ±5
Average Score (All Tasks)	1	30	38	–	49	40	44	64	**66**
Average Score (Hard Tasks)	0	8	21	–	30	20	21	45	**50**

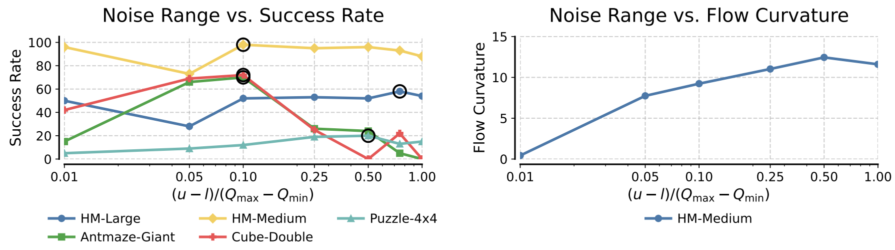

Figure 12: ***Effect of variance of the initial noise sampling distribution on `floq`.*** **Left:** Success rates across environments as a function of the initial noise scaling factor (black circles denote the best setting per environment). **Right:** Flow curvature in HM-Medium increases with noise variance, highlighting the tradeoff between too little curvature (flow collapses to monolithic critic) and too much curvature (difficult numerical integration).

A.4 Additional Ablation Studies for `floq`

1) Ablations for the width of the $[l, u]$ interval. We study the effect of varying the variance of the initial noise sample used in critic flow matching by expanding the width $u-l$ of the interval that the initial noise is sampled from. We present the results in Figure 12. On the left, we observe that the performance across several tasks typically peaks at intermediate variance values (note that the black circles marking the setting that yields the best success rate for each environment). This means that choosing an interval $[l, u]$ with a non-trivial width is important. As discussed in Section 4.3, Figure 12 (right) shows that the curvature of the learned flow increases as the width of the interval grows. We measure curvature by computing the magnitude of the derivative of the velocity field as a function of time using finite differences. Concretely, we measured the expected value of $|{}^{dv_\theta(t, \boldsymbol{z}(t))}/_{dt}|$ across state-action pairs in the offline dataset, averaged through training.

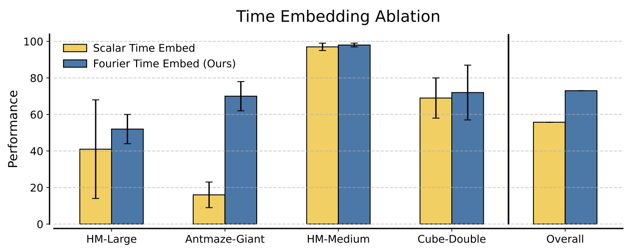

Figure 11: ***Time embedding.*** Replacing the Fourier-basis embedding of time with a scalar embedding results in significantly worse performance, highlighting the importance of Fourier features for conditioning on time.

Putting results in Figure 12 together, we note that some degree of curvature is necessary for best performance, which is expected because otherwise, the flow collapses to behave like a monolithic critic. That said, excessive curvature makes the flow numerically harder to integrate, ultimately degrading performance. Based on these observations, we recommend practitioners use $\kappa := (u - l)/(Q_{\max} - Q_{\min})$ in the range of $\{0.1, 0.25\}$ as reliable starting points when tuning `floq` critic.

2) Ablations for the time embedding. In the default configuration of `floq`, we used a 64-dimensional Fourier embedding for the time t, provided as input to the velocity field (also see Dasari et. al [11] for a recent work training a diffusion policy also using Fourier embedding of t). As shown in Figure 11,

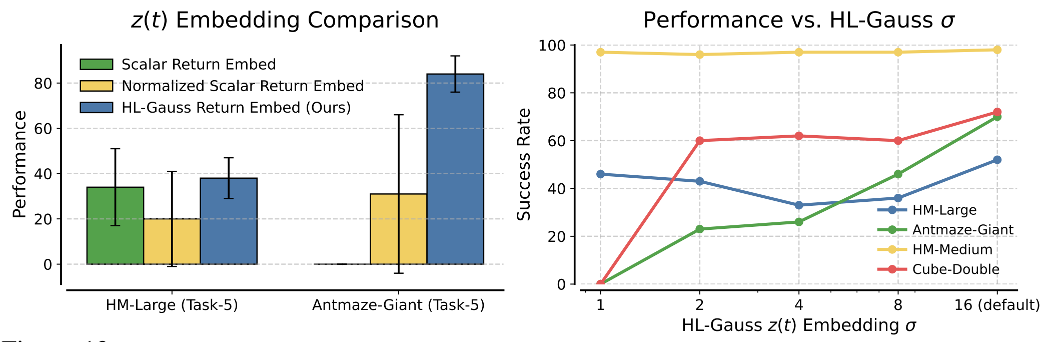

Figure 13: ***Comparison of different approaches for representing the input interpolant in*** `floq`. **Left**: performance on two representative tasks where that HL-Gauss embeddings outperform scalar and normalized scalar embeddings by reducing sensitivity to non-stationary inputs. **Right:** Ablation over HL-Gauss embedding scale σ for the scalar flow interpolant input, showing that larger values provide broader bin coverage and stronger performance. Default $\sigma = 16.0$.

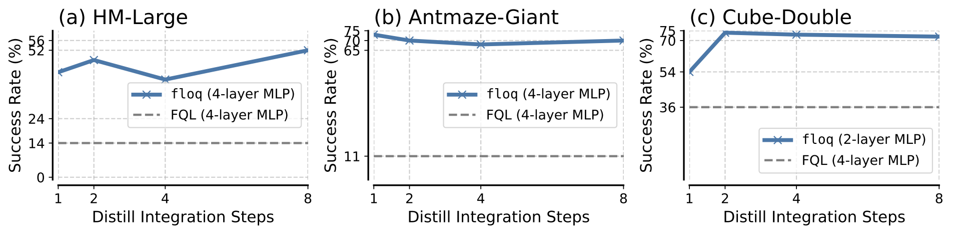

Figure 14: ***Effect of the number of integration steps used for policy extraction on*** `floq` ***performance.*** Even though computing the target values for TD-learning utilizes a fixed number of 8 integration steps, in this ablation we utilize a smaller number of steps for extracting the policy. Performance is more robust to the number of integration steps used for policy extraction, suggesting that as long as target integration is sufficiently accurate, few steps suffice for policy distillation.

replacing this Fourier embedding with a simple scalar embedding of t leads to a significant drop in performance on several tasks. This highlights the importance of the Fourier embedding, which allow the velocity function to be meaningfully conditioned on t, enabling it to produce distinct behaviors at different integration times. Without such rich embeddings, the critic struggles to leverage temporal information effectively, and again collapse to the monolithic architecture. We therefore recommend that practitioners carefully utilize high-dimensional embeddings of time when using `floq`.

3) How does the approach of embedding the interpolant $z(t)$ affect `floq` ***performance?*** We observe that the approach of embedding $z(t)$ (Design Choice 2 in Section 4.3) plays a significant role in the performance of `floq`. As shown in Figure 13, HL-Gauss embeddings of $z(t)$ provide a significant advantage over scalar or normalized scalar embeddings. In particular, across several tasks we found HL-Gauss embeddings (with a sufficiently large value of σ) to be essential for achieving strong performance, and Figure 13 (left) highlights two representative tasks in this category. HL-Gauss embeddings with broader bin coverage helps reduce the sensitivity of the network to non-stationary inputs, thereby stabilizing training and improving performance. While normalizing $z(t)$ helps on some tasks over using the raw value, we found that HL-Gauss embeddings generally gave the best performance. In our implementation of the velocity network, we use HL-Gauss embeddings with a default scale of $\sigma = 16.0$. Figure 13 (right) shows an ablation over smaller values of $\sigma \in \{1.0, 2.0, 4.0, 8.0\}$. Observe that larger values of σ consistently yield stronger performance. Intuitively, increasing σ leads to broader bin coverage for the HL-Gauss distribution (see Figure 2, right), which helps mitigate the non-stationarity of the range of $z(t)$ over the course of training with TD-learning. These results highlight that selecting sufficiently large embedding scales is important for stabilizing learning and achieving strong downstream performance.

4) How does the number of critic flow steps used for the policy update affect the performance of `floq`***?*** We next investigate the effect of varying the number of integration steps used for calculating the Q-value for the policy update. Since we build our algorithm on top of FQL, we implement the policy update by first distilling the values produced by the flow critic into a one-step, monolithic Q-function. Then the policy extraction procedure (akin to SAC+BC) maximize the values of this distilled critic subject to a behavioral cloning loss. Note that this approach essentially decouples the number of integration steps used to compute the TD-target and the number of integration steps for policy extraction. As shown in Figure 14, as long as the number of integration steps for computing the target value are fixed (to 8 in this case), the performance of `floq` is relatively robust to the number of integration steps used for the policy update. Contrast this with the sensitivity to the number of integration steps used for computing the TD-target observed in Figure 4. The results indicate that once the target integration steps are sufficiently large (here, 8), the policy can be effectively distilled even with a small number of integration steps.

A.5 Hyperparameters and Additional Details

In this section, we present some details for `floq` that we could not cover in the main paper, along with a pseudocode and a complete list of hyperparameters used by our approach.

Algorithm 1 Critic Flow Matching (`floq`) in conjunction with FQL [57]

```
Given: offline dataset of transitions $\mathcal{D}$,
Models: a flow critic, $Q_\theta^{\text{FLOW}}(s, a, \boldsymbol{z})$, a distilled critic, $Q_\psi^{\text{distilled}}(s, a, \boldsymbol{z})$, a flow policy $\pi_\phi(\cdot|s)$, one-step policy $\mu_\omega(s, \cdot)$.
function $Q_\theta^{\text{FLOW}}(s, a, \boldsymbol{z})$                         ▷ Flow Q-function, introduced by floq
    for $t = 0, 1, \dots, K-1$ do
        $\boldsymbol{z}(t+1) \leftarrow \boldsymbol{z}(t) + 1/K \cdot v_\theta\left(t/K, \boldsymbol{z}(t) \mid s, a\right)$        ▷ Euler method, time $t$ is normalized
    return $\boldsymbol{z}(K)$
function $\pi_\phi(a|s)$          ▷ Flow policy from FQL, though policy training is orthogonal to floq
    for $t = 0, 1, \dots, M-1$ do
        Sample $\boldsymbol{x}(0) \sim \mathcal{N}(0, I_d)$
        $\boldsymbol{x}(t+1) \leftarrow \boldsymbol{x}(t) + 1/M \cdot w_\phi\left(t/M, \boldsymbol{x}(t) \mid s\right)$        ▷ Euler method, time t is normalized
    return $\boldsymbol{x}(M)$
while not converged do
    Sample batch $\{(s, a, r, s')\} \sim \mathcal{D}$
    ▷ Train vector field $v_\theta$ in flow critic $Q_\theta^{\text{FLOW}}$
    $a' \leftarrow$ Sample$(\pi_\phi(\cdot|s'))$        ▷ Sample actions from policy, typically the one-step policy for FQL
    $\boldsymbol{z}(0) \sim$ Unif $[l, u]$, $\boldsymbol{z}'(0)_{1:m} \sim$ Unif $[l, u]$        ▷ Sample initial noise for computing the Q-value
    $\boldsymbol{z}(1) \leftarrow r + \gamma \cdot 1/m \cdot \sum_{i=1}^m Q_\theta^{\text{FLOW}}(s', a', \boldsymbol{z}_i'(0))$        ▷ Use noise $\boldsymbol{z}_i'(0)$ for computing TD-target
    $\boldsymbol{z}(t) \leftarrow (1-t) \cdot \boldsymbol{z}(0) + t \cdot \boldsymbol{z}(1)$        ▷ Compute interpolant $\boldsymbol{z}(t)$ for random $t$
    Update $\theta$ to minimize $\mathbb{E}\left[\left(v_\theta\left(t, \boldsymbol{z}(t) \mid s, a\right) - \left(\boldsymbol{z}(1) - \boldsymbol{z}(0)\right)\right)^2\right]$        ▷ Linear flow-matching loss
    ▷ Train distill critic $Q_\psi^{\text{distill}}$ for policy extraction
    Update $\psi$ to minimize $\mathbb{E}_{\boldsymbol{z}(0)}\left[(Q_\psi^{\text{distill}}(s, a) - Q_\theta^{\text{FLOW}}(s, a, \boldsymbol{z}(0)))^2\right]$
    ▷ Train a BC flow policy $\pi_\phi$, analogous to FQL
    $\boldsymbol{x}(0) \sim \mathcal{N}(0, I_d)$
    $\boldsymbol{x}(1) \leftarrow a$
    $t \sim$ Unif$([0, 1])$
    $\boldsymbol{x}(t) \leftarrow (1-t) \cdot \boldsymbol{x}(0) + t \cdot \boldsymbol{x}(1)$        ▷ For FQL policy, compute policy interpolant
    Update $\phi$ to minimize $\mathbb{E}\left[\|w_\phi\left(t, \boldsymbol{x}(t)|s\right) - \left(\boldsymbol{x}(1) - \boldsymbol{x}(0)\right)\|_2^2\right]$        ▷ Flow-matching loss for policy
    ▷ Train one-step policy $\mu_\omega$ to maximize the learned distill critic while staying close to BC flow policy
    $\boldsymbol{x} \sim \mathcal{N}(0, I_d)$
    $a^\pi \leftarrow \mu_\omega(s, \boldsymbol{x})$
    Update $\omega$ to minimize $\mathbb{E}\left[-Q_\psi^{\text{distill}}(s, a^\pi) + \alpha\|a^\pi - \pi_\phi(s, z)\|_2^2\right]$
return One-step policy $\pi_\omega$
```

Efficient policy extraction using a distilled critic. Because `floq` parameterizes a flow-matching critic, extracting reparameterized policy gradients requires computing gradients of the full integration process with respect to the input action, which is a costly operation especially when using many integration steps. To reduce this overhead, we adapt a technique introduced by Park et al. [57] for flow-matching policies and apply it to critics. Specifically, we train a *distilled critic*, $Q_\psi^{\text{distill}}(s, a)$, to approximate the predictions obtained by integrating the flow critic, $Q_\theta^{\text{flow}}(s, a, \boldsymbol{z})$. Policy extraction is then performed directly on the distilled critic. Importantly, the distilled critic is not conditioned on the noise $\boldsymbol{z}$, since in practice we only use the mean prediction of the flow critic. This design allows the distilled critic to implicitly capture that behavior while eliminating unnecessary conditioning. We illustrate this idea in Algorithm 1.

Hyperparameters for offline RL results. Following Park et al. [57], we tune the BC coefficient α on the `default-task` of each environment and then fix this value for the remaining tasks. For both FQL and `floq`, α is tuned over $\{\alpha_{\text{FQL}} - \Delta,\ \alpha_{\text{FQL}},\ \alpha_{\text{FQL}} + \Delta\}$, where $\Delta = 100$ for the `puzzle`, `cube`, and `scene` environments, and $\Delta = 10$ for the `ant` and `humanoid` environments. The baseline values α_{FQL} are taken from Table 6 in Park et al. [57], and the final values for both methods are

Table 3: **Hyperparameters for `floq`.** Differences from FQL are shown in light blue within brackets. Other hyperparameters are kept to be the same as FQL.

Hyperparameter	**Value** (`floq`)
Learning rate	0.0003
Optimizer	Adam (Kingma & Ba, 2015 [32])
Gradient steps	2M (Offline), 1M + 2M (Online FT)
Minibatch size	256 (default), 512 for `hm-large`, `antmaze-giant`
Flow Q Network MLP dims	[512,512,512,512] (default), [512,512] for `cube` envs
Distill Q MLP dims	[512,512,512,512] (not used in FQL)
Nonlinearity	GELU (Hendrycks & Gimpel, 2016 [26])
Target network smoothing coeff.	0.005
Discount factor γ	0.99 (default), 0.995 for `antmaze-giant`, `humanoidmaze`, `antsoccer`
Flow time sampling distribution	Unif($[0, 1]$)
Clipped double Q-learning	False (default), True (`antmaze-giant`) (+ `antmaze-large` in FQL)
BC coefficient α	Tables 4, 5
Actor Flow steps	10
Critic Flow steps	8 (default), Table 6 for env-wise (not used in FQL)
Initial Sample Range	0.1 (default), Table 6 for env-wise (not used in FQL)
Number Of Initial Noise Samples	8 (not used in FQL)
Fourier Time Embed Dimension	64 (not used in FQL)

Table 4: Environment-wise BC-Coefficient (α) for FQL and `floq` (Offline RL).

`Environment (5 tasks each)`	α **(FQL),** α **(**`floq`**)**
`antmaze-large`	$10, 10$
`antmaze-giant`	$10, 10$
`hmmaze-medium`	$30, 30$
`hmmaze-large`	$30, 20$
`antsoccer-arena`	$10, 10$
`cube-single`	$300, 300$
`cube-double`	$300, 300$
`scene-play`	$300, 300$
`puzzle-3x3`	$1000, 1000$
`puzzle-4x4`	$1000, 1000$

reported in Table 4. For `floq`, after tuning α with the default configuration ($K = 8$ flow steps and width $(u - l) = \kappa(Q_{\max} - Q_{\min})$ with $\kappa = 0.1$), we tune $K \in \{4, 8\}$ and $\kappa \in \{0.1, 0.25\}$ on the `default-task` of each environment. These values, referred to as `floq`(Best), are reported in Table 6. In all cases, for `floq`, we utilize $m = 8$ samples of initial noise to compute the target Q-value as discussed in Section 4.2.

Hyperparameters for online fine-tuning. Most hyperparameters (unless otherwise stated) remain similar in online fine-tuning and offline RL pre-training. For both FQL and `floq`, α is tuned in the range $[10, 100]$ (step size 10) for the `ant` and `humanoid` environments, and in $[100, 1000]$ (step size 100) for the `cube`, `scene`, and `puzzle` environments. The selected α values are given in Table 5.

For `floq`, after tuning α with the default configuration ($K = 8$, $\kappa = 0.1$), we tune $K \in \{4, 8, 16\}$ and $\kappa \in \{0.1, 0.25\}$ per environment. The chosen values are reported in Table 7.

Number of seeds. We ran 3 seeds for each configuration of both `floq` and **FQL** on each task, for both offline RL and online fine-tuning.

In summary, we tuned the common hyperparameters for `floq`(Def.) and **FQL** the same amount on the default task for each environment (following [56]). For `floq`(Best), we additionally tuned the `floq` specific hyper-parameters K and κ on the default task.

Table 5: Environment-wise BC-Coefficient (α) for FQL and `floq` (Online Fine-Tuning).

`Environment (5 tasks each)`	α **(FQL),** α **(**`floq`**)**
`antmaze-large`	$10, 10$
`antmaze-giant`	$10, 10$
`hmmaze-medium`	$80, 30$
`hmmaze-large`	$40, 20$
`antsoccer-arena`	$30, 30$
`cube-single`	$300, 300$
`cube-double`	$300, 300$
`scene-play`	$300, 300$
`puzzle-3x3`	$1000, 1000$
`puzzle-4x4`	$1000, 1000$

Table 6: Environment-wise Initial Sample Range ($\frac{u-l}{Q_{\max}-Q_{\min}}$) and Flow Steps ($K$) for `floq` (Best) (Offline RL).

`Environment (5 tasks each)`	$(\frac{u-l}{Q_{\max}-Q_{\min}}, K)$
`antmaze-large`	$(0.1, 8)$
`antmaze-giant`	$(0.1, 4)$
`hmmaze-medium`	$(0.1, 8)$
`hmmaze-large`	$(0.1, 8)$
`antsoccer-arena`	$(0.1, 8)$
`cube-single`	$(0.1, 8)$
`cube-double`	$(0.1, 8)$
`scene-play`	$(0.1, 8)$
`puzzle-3x3`	$(0.1, 8)$
`puzzle-4x4`	$(0.25, 8)$

Table 7: Environment-wise Initial Sample Range ($\frac{u-l}{Q_{\max}-Q_{\min}}$) and Flow Steps ($K$) for `floq` (Online FT).

`Environment (5 tasks each)`	$(\frac{u-l}{Q_{\max}-Q_{\min}}, K)$
`antmaze-large`	$(0.1, 8)$
`antmaze-giant`	$(0.1, 4)$
`hmmaze-medium`	$(0.1, 8)$
`hmmaze-large`	$(0.1, 16)$
`antsoccer-arena`	$(0.1, 8)$
`cube-single`	$(0.1, 8)$
`cube-double`	$(0.1, 8)$
`scene-play`	$(0.1, 8)$
`puzzle-3x3`	$(0.1, 8)$
`puzzle-4x4`	$(0.25, 8)$

A.6 Flow Visualizations

We visualize the evolution of the learned flow critic during training on `cube-double` in Figure 15, with $\kappa = 0.1$. Because raw Q-values can have large magnitudes, directly plotting them makes it difficult to assess the curvature of the learned flow. Instead, we plot advantage values, defined as the gap between the predicted Q-value obtained by integrating for k flow steps at various noise samples $\boldsymbol{z}_i(0)$, namely $\psi(k, \boldsymbol{z}_i(k) \mid s, a)$ for $i \in [5], , k \in [1, \dots, K]$, and the expected value of that state–action pair after K steps, scaled linearly to k steps. Put simply, this advantage quantifies how far the intermediate estimate $\psi(k, \boldsymbol{z}_i(k) \mid s, a)$ deviates from the "straight line" path between the initial noise sample $\boldsymbol{z}_i(0)$ and the final Q-value. We find that these deviations are consistently non-zero and vary substantially across the integration process. In many cases, they exhibit a characteristic pattern of overshooting followed by correction: larger deviations early on that diminish as integration proceeds. These dynamics provide direct evidence that the learned flows follow curved rather than

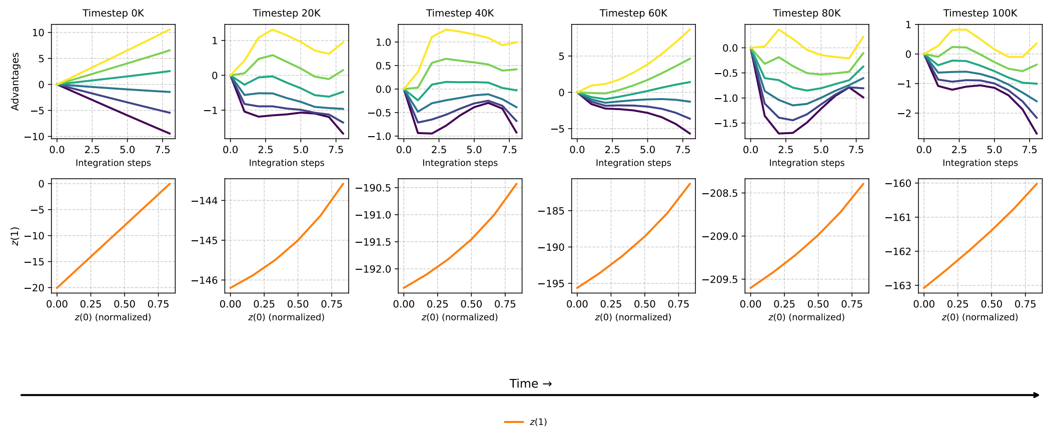

Figure 15: ***Visualizing the evolution of the trajectories of the flow critic during training.***

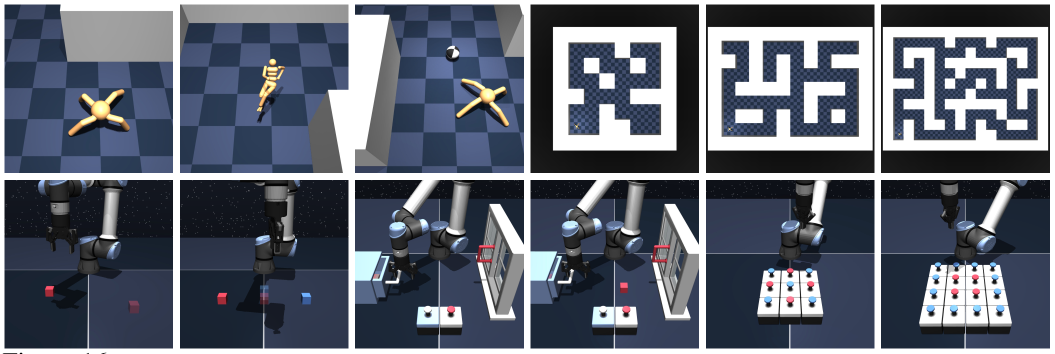

Figure 16: **OGBench [54] domains**. These tasks include high-dimensional state and action spaces, sparse rewards, stochasticity, as well as hierarchical structure.

linear trajectories. We also visualize the final Q-value output $z(1)$ as a function of the input $z(0)$ in Figure 15 (bottom) and find that the final $z(1)$ depends non-linearly on the initial noise value.

A.7 Environment Visualizations

We visualize OGBench tasks in Figure 16.

A.8 Wall Clock Run-Time

We report the wall clock run-times for FQL and `floq` in Table 8.

Table 8: Total wall-clock runtime (in 10^3 seconds) for FQL and `floq` with varying numbers of flow integration steps across four representative environments. Reported numbers correspond to 2M training steps.

Environment	FQL	`floq` (Flow Steps)				
		1	2	4	8	16
HM-Maze Large	14	24	28	35	50	79
HM-Maze Medium	12	19	21	23	30	47
Cube-Double	10	15	16	17	19	26
Antmaze-Giant	10	20	24	30	45	74