WHAT MATTERS FOR VALUE GUIDANCE OF FLOW POLICIES IN REINFORCEMENT LEARNING?

Anonymous authors
Paper under double-blind review

ABSTRACT

Flow policies are increasingly used to model behavior policies in offline reinforcement learning, but their iterative generative process complicates value-based policy improvement. Value guidance offers an alternative that leaves the behavior policy parameters unchanged by steering the generation process with the gradient of a learned critic. Existing value guidance methods instantiate this idea in diverse ways, differing in the flow timestep at which guidance is applied, how many guidance steps are taken, and how the guidance signal is constructed from the critic. Because these factors are rarely varied or studied in isolation, it is unclear which of them drive performance. In this work, we first identify which of these factors matter through a controlled study that varies each independently, and then examine how the findings change when the critic is trained on actions produced by the guidance itself. We find that the guidance signal determines which flow timesteps admit effective guidance, while the benefit of additional guidance steps differs across tasks. A single guidance step often suffices in tasks where the behavior cloning (BC) policy already performs well, whereas more steps are needed to substantially improve performance in tasks where the BC policy rarely succeeds. We also find that when the critic learns from bootstrap actions generated by the guidance itself, the benefit of additional guidance steps saturates earlier, with only a few steps recovering most of the attainable improvement. Building on this, we show that applying only a few guidance steps at selected flow timesteps matches or exceeds guiding at every flow step while reducing inference cost by up to 63%, showing that these insights translate into practical and efficient value guidance. These findings suggest that guidance schedules and value learning should be designed together to exploit the iterative structure of flow policies for effective value guidance.

1 INTRODUCTION

Reinforcement learning (RL) seeks to learn effective decision-making policies through reward feedback, often using a learned value function to guide policy optimization. Following advances in generative modeling, diffusion and flow models (Ho et al., 2020; Song et al., 2021; Liu et al., 2023; Lipman et al., 2023) have increasingly been adopted as policy parameterizations for their ability to capture complex behavior distributions via an iterative generative process (Wang et al., 2023; Hansen-Estruch et al., 2023; Lyu et al., 2025; Park et al., 2025b). However, this iterative process complicates value-based optimization since directly optimizing generated actions through the policy requires backpropagation through the sampling chain, which can be unstable and computationally expensive. This has motivated approaches that enable action sampling in fewer generation steps (Kang et al., 2023; Ding & Jin, 2024; Park et al., 2025b), learning value functions over noisy actions (Doo et al., 2026; Selim et al., 2026), or adopting specialized objectives such as adjoint matching (Domingo-Enrich et al., 2025; Li & Levine, 2026; Dong et al., 2026b), all of which update the policy parameters directly to maximize value.

A second line of work instead leaves the pretrained behavior cloning (BC) policy untouched and steers the sampling process with the gradient of the critic at test time. This allows it to improve actions while preserving the behavior the BC prior has learned from demonstrations, making it especially attractive in robotic manipulation. Because flow policies generate actions iteratively, guidance can intervene at different points during this process: some prior methods steer the initial noise, refine the final action sample, or intervene throughout generation (Wagenmaker et al., 2025; Simsir & Oguz,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2026; Johannink et al., 2019; Dong et al., 2026a; Zhou et al., 2026; Su et al., 2026). Although these methods differ substantially, they raise a shared question: which parts of the generation process benefit from value-based intervention, and how much intervention is needed? Without knowing the effects of each component, it is difficult to improve these methods systematically. Separating their effects can reveal how value information should be used during action generation and where additional guidance computation is actually useful.

Answering this question from comparisons between existing complete algorithms is difficult because the algorithms also differ in how many guidance steps are applied, and how the guidance signal is constructed. In this work, we view these methods within a unified framework that expresses value guidance through three potential contributing factors, allowing us to vary each independently and understand what drives the improvements of value guidance: (1) the flow *timestep* at which guidance is applied, (2) the number of guidance steps, and (3) how the guidance signal is computed from the same critic. We begin with a controlled study in which different guidance methods share the same pretrained critic and BC flow policy, with neither network receiving updates from the guidance strategy. Within this setup, we independently vary those three factors while fixing the the total amount of guidance applied, measured as the sum of magnitude of the guidance updates, which we call the guidance *budget*. We then extend this study and see whether these findings hold under *guided bootstrapping*, where the critic learns from actions produced by guidance itself.

Our controlled study shows that the guidance signal determines which timesteps are effective for guidance. Specifically, a signal based on noisy-space value function supports guidance across a broader range of timesteps, whereas one-step signals perform best in later timesteps. At effective guidance timesteps, we find that the role of the number of guidance steps differs across tasks. While single guidance step often suffices in task where BC alone is a suitable objective, dividing the same budget across more steps substantially improves performance in tasks where the BC policy rarely solves the task and requires stronger guidance. Furthermore, on these harder tasks, we find that guided bootstrapping raises performance overall while reaching near-best performance with only a few guidance steps.

Together, our findings show that effective value guidance requires a guidance signal that remains accurate on intermediate flow timesteps, a sufficient number of guidance steps at timesteps where that signal is reliable, and a critic trained on the actions that guidance produces. Building on this, we evaluate guidance schedules that apply only a few guidance steps at selected flow timesteps to test whether these insights translate into practical and efficient value guidance. In our experiments, these schedules match or exceed guiding at every flow step while reducing inference cost by up to 63%, and also outperform recent competitive baselines. These findings suggest that guidance schedules and value learning should be designed together to exploit the iterative structure of flow policies, opening the way toward value guidance that is both efficient and effective.

2 Related Work

Offline reinforcement learning. Offline RL aims to learn a policy from a fixed dataset without additional environment interaction (Levine et al., 2020). A central challenge is distributional shift, as value estimates may be unreliable on actions that are poorly represented in the dataset. Existing methods address this issue by constraining the policy to remain close to the dataset actions (Peng et al., 2019; Wu et al., 2019; Nair et al., 2020; Fujimoto & Gu, 2021; Chen et al., 2023; Tarasov et al., 2023). Complementary approaches address this issue via penalizing out-of-distribution (OOD) actions, using an ensemble of critic networks, or maximizing the value within the support of the offline dataset. (Fujimoto et al., 2018; 2019; Kumar et al., 2020; Kostrikov et al., 2021; An et al., 2021).

Generative policies and value guidance. Diffusion and flow models capture complex distributions through iterative generative processes (Ho et al., 2020; Song et al., 2021; Liu et al., 2023; Lipman et al., 2023), motivating their use as expressive policy representations. Their use in RL spans several policy-learning formulations, including value-weighted behavioral learning (Hansen-Estruch et al., 2023; Chen et al., 2023), direct critic-based policy optimization (Wang et al., 2023; Kang et al., 2023; Ding & Jin, 2024), distillation-based approaches (Park et al., 2025b), and objectives based on adjoint matching (Li & Levine, 2026; Dong et al., 2026b). In generative modeling, guidance

steers samples toward high-value outcomes using an auxiliary signal, such as a classifier (Dhariwal & Nichol, 2021; Bansal et al., 2024; Feng et al., 2025; Liu et al., 2025). In RL, this guidance signal is often derived from the gradient of a learned critic network, and existing methods differ in how this signal is constructed from the critic. The critic can be evaluated directly at the partially denoised action (Psenka et al., 2024; Fang et al., 2025), at a one-step predicted action (Zhou et al., 2026; Wang et al., 2026), or through a noisy-space value function that learns the value of partially denoised actions (Doo et al., 2026; Selim et al., 2026).

3 Preliminaries

Offline reinforcement learning. We consider a Markov decision process $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A} \subseteq \mathbb{R}^{d_a}$ is the action space, $P(s' \mid s, a)$ is the transition distribution, $r(s, a)$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. We use n to index environment steps, reserving t for flow timesteps.

In offline RL, a policy $\pi(a \mid s)$ is learned from a fixed dataset $\mathcal{D}$ of transitions (s, a, r, s'). Its action-value function is the expected discounted return after taking action a in state s,

$$Q^\pi(s, a) = \mathbb{E}_\pi\Big[\textstyle\sum_{n=0}^{\infty} \gamma^n r(s_n, a_n) \,\Big|\, s_0 = s,\ a_0 = a\Big],$$

which a critic network Q_ϕ often learns via temporal-difference (TD) learning:

$$\mathcal{L}_Q(\phi) = \mathbb{E}_{\substack{(s,a,r,s')\sim\mathcal{D} \\ a'\sim\pi_\theta(\cdot|s')}}\left[\left(Q_\phi(s, a) - r - \gamma Q_{\bar{\phi}}(s', a')\right)^2\right], \tag{1}$$

where $\bar{\phi}$ is an exponential moving average of ϕ (Mnih et al., 2013). Since the critic is unreliable for actions poorly covered by $\mathcal{D}$, most methods update a policy to maximize Q_ϕ while penalizing divergence from the behavior policy. With action chunking, the policy outputs a sequence of low-level actions, which we treat as a single chunked action $a \in \mathcal{A}$ for value learning (Li et al., 2025).

Flow matching and flow policies. A flow policy generates an action by transporting noise $a_0 \sim \mathcal{N}(0, I_{d_a})$ to an action a_1 along a state-conditioned velocity field $v_\theta(s, a_t, t)$, where $t \in [0, 1]$ denotes the flow timestep. Flow matching trains the velocity field with

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{\substack{(s,a)\sim\mathcal{D},\ a_0\sim\mathcal{N}(0,I_{d_a}) \\ t\sim\mathcal{U}[0,1]}}\left[\|v_\theta(s, a_t, t) - (a - a_0)\|_2^2\right], \tag{2}$$

where $a_t = (1 - t)a_0 + ta$ is the linear interpolant. Trained on $\mathcal{D}$, this objective yields a BC flow policy; we denote the resulting velocity field and policy by v_{BC} and π_{BC}, respectively. Actions are sampled by integrating v_{BC} from $t = 0$ to 1 with N Euler steps:

$$a_{t+1/N} = a_t + \tfrac{1}{N}\, v_{\mathrm{BC}}(s, a_t, t), \qquad t \in \{0, \tfrac{1}{N}, \ldots, \tfrac{N-1}{N}\}. \tag{3}$$

Value guidance. Value guidance steers the BC sampling process toward high-value actions by adding a guidance signal $g(s, a_t, t)$ to the BC velocity field at each Euler step:

$$a_{t+1/N} = a_t + \tfrac{1}{N}\left[v_{\mathrm{BC}}(s, a_t, t) + \alpha\, g(s, a_t, t)\right], \tag{4}$$

where $\alpha \geq 0$ controls the guidance strength and $\alpha = 0$ recovers Eq. (3). The common practice is to apply guidance at every one of the N Euler steps, which we refer to as *dense guidance*. Applying it only at a subset of these steps is what we refer to as *sparse guidance* and is less explored. The simplest guidance signal is the critic's action gradient evaluated at the partially denoised action a_t: $g(s, a_t, t) = \nabla_{a_t} Q_\phi(s, a_t)$ (Psenka et al., 2024; Fang et al., 2025). However, since the critic is trained only on "clean" actions, this evaluation is essentially OOD. In this work, we consider three alternative signals that avoid this issue in Section 4.1.

4 Understanding Value Guidance

To better understand the workings of guidance approaches, we investigate the effects of three potential contributing factors: *timestep* **(T)**, the flow time t at which guidance is applied; *iterations* **(I)**, the

Table 1: **Guidance signals (S) compared in this work.**

Method	**Guidance signal** $g(s, a_t, t)$	**Additional critic?**	**Gradient through** v_{BC}**?**
Noisy-space critic (Doo et al., 2026)	$\nabla_{a_t} Q_\omega(s, a_t, t)$	✓	×
One-step QGF (Zhou et al., 2026)	$\nabla_{\hat{a}_1} Q_\phi(s, \hat{a}_1)$	×	×
One-step reparameterized	$\nabla_{a_t} Q_\phi(s, \hat{a}_1)$	×	✓

number of guidance steps taken; and *signal* **(S)**, how the guidance direction $g(s, a_t, t)$ is obtained from the critic. Because these factors are rarely varied in isolation, their individual effects are often difficult to distinguish in existing performance comparisons of prior methods. To better isolate and study these effects, we first pretrain a BC flow policy and a critic trained independently of any guidance strategy. We then study the impact of different choices of **(T)**, **(I)**, and **(S)** on these pretrained models. Our experiments reveal that the choice of **(S)** strongly shapes which choices of **(T)** are effective, while the benefits of **(I)** are environment-dependent.

4.1 Analysis setup

Value learning objective. To ensure fair comparisons between different guidance techniques, we choose a value learning objective that does not depend on the guidance strategy. Specifically, we use Expected-Max Q-learning (EMaQ; Ghasemipour et al. 2021), which estimates an off-policy value target via Best-of-N (BoN) sampling with π_{BC}:

$$\mathcal{L}(\phi) = \mathbb{E}_{\substack{(s,a,r,s')\sim\mathcal{D} \\ \{a'_{(i)}\}_{i=1}^{M}\sim\pi_{\mathrm{BC}}(\cdot|s')}} \left[\left(Q_\phi(s, a) - r - \gamma Q_{\bar{\phi}}(s', a'^{\star}) \right)^2 \right], \tag{5}$$

where M is the number of candidates, and $a'^{\star} = \arg\max_{a' \in \{a'_{(1)}, \ldots, a'_{(M)}\}} Q_\phi(s', a')$.[1]

Guidance signals. Given the critic Q_ϕ, we compare three choices of the guidance signal **(S)**, summarized in Table 1. The *noisy-space critic* (Doo et al., 2026) learns an additional critic $Q_\omega(s, a_t, t) \approx Q_\phi(s, a_1)$ that operates directly on intermediate actions and predicts the value of the BC endpoint reached from a_t. Instead of learning a separate critic, the other two approximate the remaining BC rollout with a single Euler step, $\hat{a}_1 = a_t + (1 - t)\, v_{\mathrm{BC}}(s, a_t, t)$, and evaluate Q_ϕ at this predicted endpoint, differing only in whether the gradient is propagated through v_{BC}; we refer to these as *one-step* signals. Further details are given in Appendix A.

Localized guidance. Guidance is typically applied at every flow step, which we refer to as dense guidance (Eq. (4)). To vary the two factors **(T)** and **(I)** independently, we study *localized guidance*, a form of sparse guidance that applies all K guidance steps at a single flow timestep t. This allows us to vary **(T)** while keeping **(I)** fixed. Starting from $a_0 \sim \mathcal{N}(0, I_{d_a})$, we follow the BC flow (Eq. (3)) until time t, at which point the intermediate action a_t is steered with K guidance steps:

$$a_t^{(k+1)} = a_t^{(k)} + \alpha\, g(s, a_t^{(k)}, t), \qquad k = 0, \ldots, K - 1, \tag{6}$$

with $a_t^{(0)} = a_t$. Generation is then finished by following v_{BC} from $a_t^{(K)}$ to $t = 1$.

Guidance budget. Comparing different guidance configurations fairly requires them to apply the same amount of guidance, which the coefficient α alone does not ensure: with the same α, more iterations **(I)** apply more guidance in total, and the size of each update depends on the gradient magnitude, which may vary widely across tasks and signals (see Appendix B.4). We measure the amount of guidance with the Euclidean norm of the guidance updates, which directly reflects how far guidance moves the action away from the BC trajectory. We define the *guidance budget* B as the sum of the expected per-update magnitudes:

$$B := \mathbb{E}_{s\sim\mathcal{D},\, a_0\sim\mathcal{N}(0, I_{d_a})} \left[\sum_{k=0}^{K-1} \frac{\alpha \left\| g(s, a_t^{(k)}, t) \right\|_2}{\sqrt{d_a}} \right], \tag{7}$$

[1] We also used Implicit Q-Learning (IQL; Kostrikov et al. 2021) to learn the critic, but we found EMaQ to be more reliable with the limited dataset sizes we used, and IQL required datasets 10–100× larger to perform.

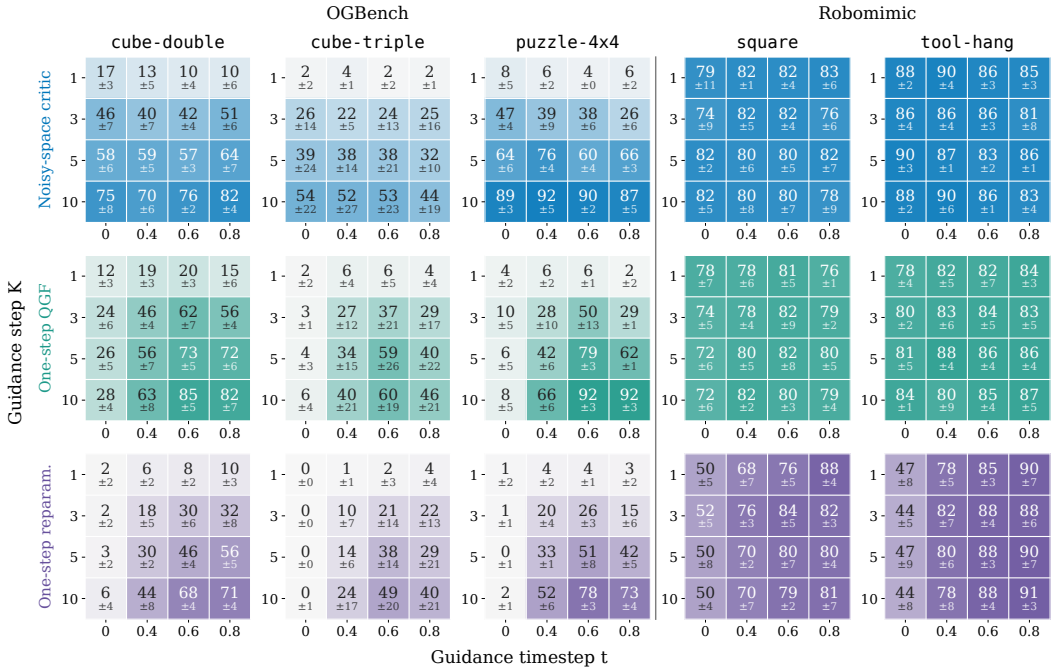


Figure 1: **Guidance scaling matrices across timestep (T) and iterations (I) for each guidance signal (S).** The effective range of **(T)** depends on **(S)**, and increasing **(I)** helps on OGBENCH but little on ROBOMIMIC. Each cell shows the success rate at the best budget (mean ± std over four seeds).

where normalizing by $\sqrt{d_a}$ makes magnitudes comparable across tasks with different action dimensionality. For each configuration, we calibrate α on 1024 dataset states until the target budget is matched and hold it fixed during evaluation.

Tasks and evaluation protocol. We evaluate on robotic manipulation tasks from two benchmarks representing different data regimes. On OGBENCH (Park et al., 2025a), we use the default tasks of `cube-double`, `cube-triple`, and `puzzle-4x4` for single-task offline RL, where the datasets contain diverse behavior. On ROBOMIMIC (Mandlekar et al., 2021), we use `square` and `tool-hang` with sparse rewards, where we collect our datasets by rolling out a BC policy trained on the original human demonstrations. All policies use action chunking; further details are listed in Appendix B.2. For each choice of signal **(S)**, we independently vary the flow timestep **(T)** $t \in \{0, 0.4, 0.6, 0.8\}$ and the number of guidance iterations **(I)** $K \in \{1, 3, 5, 10\}$, sweeping over a range of budget values B. Each configuration is evaluated over four seeds, using 50 episodes per seed. We provide further analysis details in Appendix B.4.

4.2 RESULTS: HOW DO GUIDANCE SIGNALS SCALE WITH GUIDANCE TIMESTEP AND STEPS?

Figure 1 reports the success rate for each configuration, with one matrix per guidance signal and task combination; rows vary the number of guidance steps K, and columns vary the guidance timestep t. Each cell shows the mean success rate across seeds of the best-performing budget B; $\pm$ denotes the standard deviation across seeds.

We observe that *the effective range of* ***(T)*** *strongly depends on* ***(S)*** on OGBENCH. While noisy-space guidance remains effective over a broad range of **(T)**, with little variation across columns in each matrix (top row), both one-step approximation-based signals (bottom two rows) perform poorly when **(T)** is confined to early flow timesteps, and their best-performing configurations generally occur at $t \geq 0.4$. A potential explanation is that the

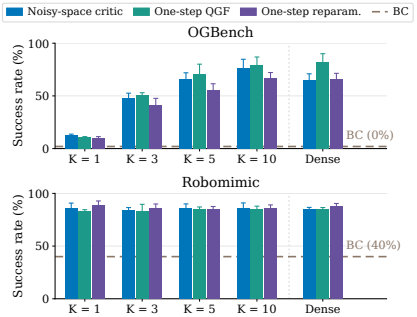


Figure 2: **Best localized guidance results by K, compared with dense guidance and unguided BC.**

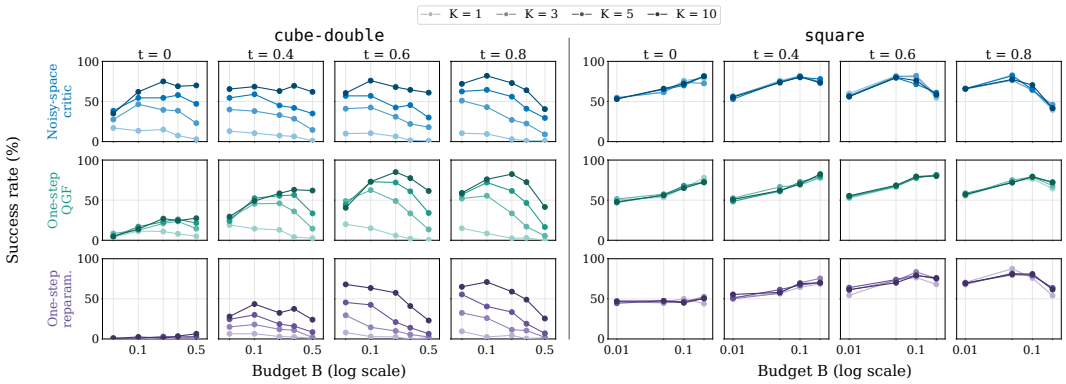


Figure 3: **Success rate across guidance budgets B for different numbers of guidance steps (I) K.** On `cube-double`, curves with larger K generally stay above those with smaller K across budgets, so increasing the budget does not substitute for additional guidance steps. On `square`, the curves largely overlap, so additional steps offer little improvement at any budget.

error in the one-step approximation of the clean action is large for early flow timesteps. Increasing **(I)** from $K = 1$ through 3, 5, and 10 yields substantial gains, although improvements are not strictly monotone in all cases.

Figure 2 summarizes this trend relative to vanilla BC and dense guidance, which applies one guidance step at every step of the generative process. Since the total budget B remains fixed, increasing **(I)** divides the same budget into more, smaller updates. The benefit of additional guidance steps is much smaller on ROBOMIMIC. Figure 2 shows that the BC baseline is already substantially stronger on `square` and `tool-hang`, and localized guidance with $K = 1$ reaches performance close to that of larger K and dense guidance. Thus, *the benefit of increasing guidance steps varies markedly by task properties*, with ROBOMIMIC exhibiting near-saturated performance with only a single guidance step.

4.3 How does the guidance budget interact with iterations?

Section 4.2 shows that increasing **(I)** substantially improves performance on OGBENCH but offers smaller gains on ROBOMIMIC. However, because Figures 1 and 2 select the budget independently for each entry, they do not reveal how this difference relates to intervention magnitude. To see whether the benefit of increasing **(I)** depends on the budget, Figure 3 compares K across budgets B at each **(T)** t on OGBENCH `cube-double` and ROBOMIMIC `square` for all three guidance signals **(S)**.

On `cube-double`, the benefit of additional guidance steps **(I)** persists across the tested budgets. At effective **(T)**, curves with larger K generally remain above those with smaller K, indicating that the scaling with **(I)** observed in Figure 1 is not merely a consequence of selecting a different budget for each configuration. Although the budget affects the overall success rate, increasing it does not substitute for increasing **(I)**.

Conversely, the curves for different K largely overlap across budgets and guidance signals **(S)** on `square`. Changing B can substantially affect performance, yet increasing **(I)** provides little consistent improvement over $K = 1$. This suggests that early saturation observed in ROBOMIMIC is not confined to a small-budget regime. Together, these results show that the trends in **(I)** observed in Figure 1 persist even when guidance budgets are compared explicitly: repeated refinement remains beneficial on `cube-double` (in which BC alone rarely solves the task), while a single guidance step is often sufficient on `square` (in which the BC policy already succeeds in a substantial fraction of episodes).

Takeaway 1: Guidance scaling depends on the guidance signal and task properties

The choice of guidance signal **(S)** determines which timesteps **(T)** are effective for sparse guidance, with noisy-space guidance supporting a broader range and one-step signals favoring later **(T)**. Within effective **(T)**, the benefit of more iterations **(I)** depends on task properties, including the performance of the underlying BC policy.

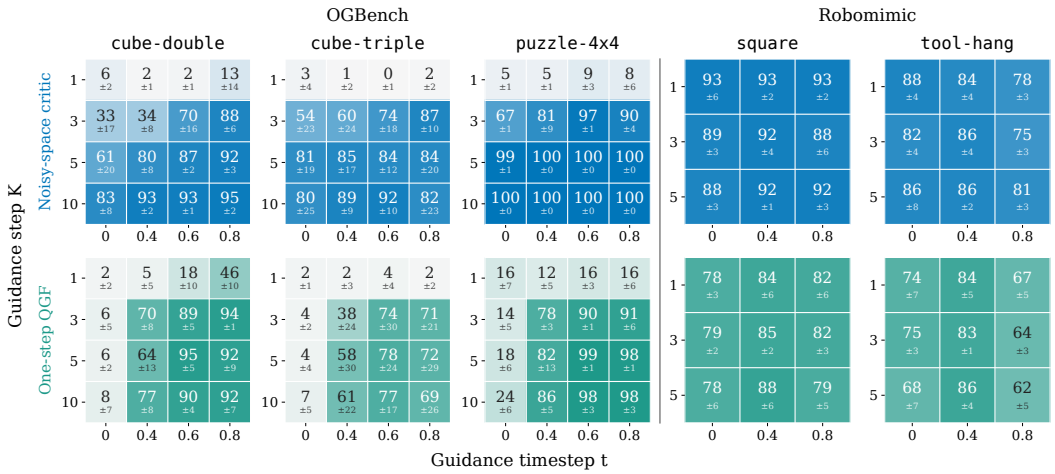


Figure 4: **Guidance scaling matrices under guided bootstrapping.** The effective range of **(T)** remains similar to that under EMaQ (Figure 1). On OGBENCH, guided bootstrapping improves performance overall and saturates after only a small **(I)**, whereas on ROBOMIMIC, performance remains similar to EMaQ. Each cell shows the success rate at the best budget (mean ± std over four seeds).

5 VALUE GUIDANCE WITH GUIDED BOOTSTRAPPING

In Section 4, we studied value guidance with a critic trained with EMaQ, independently of the guidance strategy. This separation leaves out a key property of TD learning, in which the critic evaluates the policy that generates its bootstrap actions. In this section, we instead generate bootstrap actions with each guidance strategy itself, which we call *guided bootstrapping*, and revisit how performance scales with flow timesteps **(T)** and the number of guidance steps **(I)**.

5.1 TRAINING SETUP

Guided bootstrapping. For each next state s', we sample $a'_0 \sim \mathcal{N}(0, I_{d_a})$ and apply localized guidance (Eq. 6) to obtain the guided action a'_1, which is then used as the bootstrap action a' in Eq. (1). As a result, the critic depends on the guidance configuration, so changing the guidance strategy also changes the training of the critic.

Amortized guidance field. Generating each bootstrap action requires K guidance steps, so computing critic gradients for every next state in every training batch would be costly. We therefore train an amortized guidance field e_ψ to predict the scaled guidance signal:

$$\mathcal{L}_{\text{guide}}(\psi) = \mathbb{E}_{\substack{(s,a)\sim\mathcal{D},\ a_0\sim\mathcal{N}(0,I_{d_a}) \\ t\sim\mathcal{U}[0,1]}} \Big[\big\| e_\psi(s, a_t, t) - \alpha\, g(s, a_t, t) \big\|_2^2 \Big], \tag{8}$$

where $a_t = (1-t)a_0 + ta$. The field replaces $\alpha\, g(s, a_t^{(k)}, t)$ at each guidance step of Eq. (6), both when generating bootstrap actions during training and at evaluation. In this section, we investigate both noisy-space guidance and one-step QGF as choices for signal **(S)**; we omit one-step reparameterized guidance as we found it to perform worse than one-step QGF in Section 4.2.

Maintaining the guidance budget. Because guided bootstrapping uses the guidance field throughout training, budget control must operate during learning rather than only calibrating the final policy for evaluation as in Section 4. Therefore, following Dong et al. (2026b) and Simsir & Oguz (2026), we use projected dual descent to adjust α in order to maintain the prescribed budget B throughout training. Appendix B.5 provides the update rule and implementation details.

Tasks and evaluation protocol. We use the same tasks as in Section 4 and train a separate critic and guidance field for every configuration, starting from the same pretrained BC policy. On OGBENCH, we use the same **(T)** and **(I)** values as in Section 4. On ROBOMIMIC, where Section 4.2 showed

Table 2: **Comparison across a broader set of tasks and baselines.** Average success rate (%) with $\pm$ 1 standard deviation across seeds. `NSC + dense` and `QGF + dense` refer to dense guidance with the noisy-space critic and one-step QGF, respectively, both trained with guided bootstrapping.

		OGBench							Robomimic		
		cube2	cube3	cube4	scene	puzzle3x3	puzzle4x4	Avg	square	tool-hang	Avg
Adjoint Matching	QAM (Li & Levine, 2026)	72 ±5	16 ±4	8 ±3	57 ±4	10 ±6	0 ±1	27	49 ±7	49 ±10	49
	TRQAM (Dong et al., 2026b)	**80** ±2	50 ±14	10 ±4	84 ±8	**100** ±0	**100** ±0	71	46 ±9	44 ±13	45
Post Processing	IFQL (Hansen-Estruch et al., 2023)	13 ±2	22 ±7	5 ±4	13 ±5	8 ±4	89 ±3	25	67 ±2	52 ±4	59
	DSRL (Wagenmaker et al., 2025)	78 ±3	30 ±10	5 ±3	**99** ±0	**100** ±0	93 ±5	68	78 ±3	48 ±20	63
	FEdit (Dong et al., 2026a)	37 ±8	13 ±5	6 ±3	72 ±6	**98** ±2	48 ±7	46	69 ±5	62 ±4	66
Value Guidance	NSC + dense	**83** ±2	**77** ±12	22 ±4	77 ±15	**100** ±0	**100** ±1	**77**	**91** ±1	**82** ±7	**87**
	QGF + dense	**81** ±3	64 ±17	**24** ±1	**96** ±1	**100** ±0	**99** ±0	**77**	81 ±5	76 ±4	79

that performance already saturates with few number of guidance steps, we use a reduced grid of **(T)** $t \in \{0, 0.4, 0.8\}$ and **(I)** $K \in \{1, 3, 5\}$. Each configuration is evaluated at the final training step over four seeds, using 50 episodes per seed. We provide further setup and training details in Appendix B.5.

5.2 Results: How does guided bootstrapping change guidance scaling?

Figure 4 reports the success rate of localized guidance under guided bootstrapping, with one matrix per guidance signal and task; rows vary the number of guidance steps K, and columns vary the timestep t. Each cell shows the mean success rate across seeds at the best-performing budget B, and $\pm$ denotes the standard deviation across seeds. The timestep **(T)** trends observed with EMaQ persist: the noisy-space critic remains effective across broad range of **(T)** (top row), whereas one-step QGF performs best at later ones (bottom row). However, the scaling with iterations **(I)** changes substantially on OGBench.

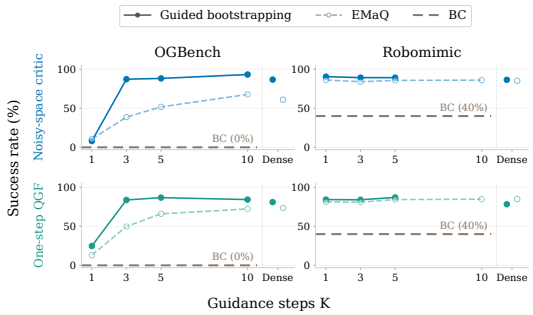


Figure 5: **Best localized performance by K.** On OGBench, guided bootstrapping performs better and saturates by $K = 3$, while on Robomimic, it performs similarly to EMaQ.

Figure 5 compares the best result for each K under guided bootstrapping (solid) and EMaQ (dashed): with $K = 3$, guided bootstrapping reaches around 95% of its best tested performance. One possible reason lies in how **(I)** enters learning. With EMaQ, K only changes how actions are extracted from a fixed critic, whereas with guided bootstrapping, it also changes the actions from which the critic's bootstrap targets are formed, so the critic is trained on the guidance it later applies. Beyond saturating earlier, guided bootstrapping also reaches higher performance on OGBench (left): at $K = 10$, it improves over EMaQ by 20 points for the noisy-space critic and 10 points for one-step QGF with localized guidance, and by 26 and 8 points with dense guidance, respectively. In contrast, on Robomimic (right), the two bootstrapping methods perform comparably across all **(I)** and under dense guidance, and a single guidance step suffices under both. This suggests that the benefit of guided bootstrapping also depends on task characteristics: when the BC policy is already strong, BoN sampling with the BC policy provides a critic of sufficient quality for value guidance.

To verify that the performance gain from guided bootstrapping is substantial, we compare dense guidance trained with guided bootstrapping against recent baselines (Table 2). The comparison covers all five tasks of six OGBench environments and both Robomimic tasks, and we provide details of the baselines and the experimental setup in Appendix B.6. On OGBench, guided bootstrapping reaches 77% on average with both guidance signals with the largest gains on the `cube` environments. On Robomimic, the gap is much larger, where guided bootstrapping reaches 87% with the noisy-space critic, 21 points above the strongest baseline, while the methods that directly update the flow policy, Q-learning with Adjoint Matching (QAM; Li & Levine 2026) and Trust Region Q-Adjoint Matching (TRQAM; Dong et al. 2026b), remain below 50%.

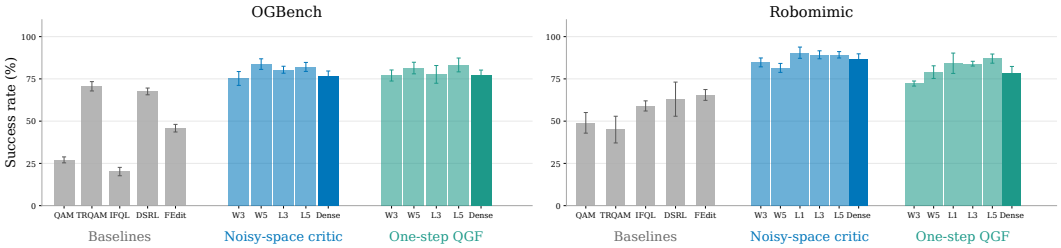


Figure 6: **Sparse guidance performance.** On the default task of each environment, sparse schedules largely retain the performance of dense guidance and outperform all baselines on both benchmarks.

5.3 Can a few guidance steps replace dense guidance?

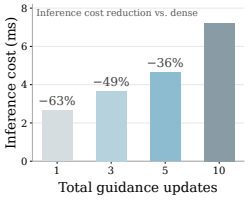


Figure 7: **Inference cost by number of guidance steps (I).**

Since guided bootstrapping retains most of the attainable performance with only a few guidance steps **(I)** (Section 5.2), efficient yet effective guidance may be possible with sparse schedules. Therefore, we consider several sparse guidance schedules, each trained with guided bootstrapping and named by its type and total number of guidance steps. *Localized* schedules apply all guidance steps at a single flow timestep **(T)** selected from Figure 4. On OGBench, we use *L3* ($t = 0.8$) and *L5* ($t = 0.6$), which we found to be the most effective per our analysis in Section 5.2. On Robomimic, where a single guidance step already suffices, we use *L1*, *L3*, and *L5* at $t = 0.4$. In practice, an effective **(T)** is unknown in advance, and finding one requires a per-task sweep. Therefore, we further consider a more practical heuristic: *window* schedules, which apply one guidance step at each of several consecutive late flow timesteps, where both guidance signals are effective: *W3* at $t \in \{0.7, 0.8, 0.9\}$ and *W5* at $t \in \{0.5, \ldots, 0.9\}$. We evaluate all schedules with both guidance signals on the default tasks of the six OGBench environments and both Robomimic tasks (see Appendix B.7).

Figures 6 and 7 report the success rate and latency of each schedule. On OGBench, all sparse schedules stay within a few points of dense guidance for both guidance signals, while using only three and five guidance steps **(I)**. On Robomimic, localized schedules match or exceed dense guidance for both signals, and a single guidance step (*L1*) already surpasses it while reducing latency by 63%. Window schedules also remain close to dense guidance on Robomimic, except for *W3* with one-step QGF, which falls several points below it. Across both benchmarks, every sparse schedule still outperforms all baselines, even when the flow timestep **(T)** is chosen by a simple heuristic rather than a per-task search. These results suggest that a sufficient number of guidance steps **(I)** at well-chosen flow timesteps **(T)** make the value guidance both effective and efficient.

Takeaway 2: Guided bootstrapping changes how guidance scales with iterations

On tasks where the BC policy alone rarely succeeds, guided bootstrapping raises overall performance and saturates with only a few guidance steps **(I)**. This is still task-dependent, since EMaQ performs comparably when the BC policy is already strong, and a single guidance step often suffices on such tasks. Consequently, a few guidance steps **(I)** at well-chosen flow timesteps **(T)** suffice to match dense guidance at a fraction of its cost.

6 Conclusion

We studied value guidance of flow policies by isolating three contributing factors: flow timestep **(T)**, iterations **(I)**, and signal **(S)**. The signal determines which flow timesteps admit useful guidance, while the benefit of more guidance steps depends on how well the BC policy already performs; when the critic learns from its own guided bootstrap actions, performance improves further and saturates with only a few guidance steps. Together, these findings suggest that effective and efficient value guidance comes not from guidance at every step, but from a sufficient number of guidance steps at well-chosen flow timesteps, with guidance schedules and value learning designed together.

AI USE STATEMENT

In this work, we used generative AI tools for designing experiments and assisting with the implementation of methods and baselines. We have not used generative AI tools for developing theoretical models or conceptual frameworks, proposing or refining hypotheses, generating synthetic datasets, cleaning or reformatting datasets, assisting with translation, or interpreting results, and formulating mathematical claims, proving mathematical claims, and supporting qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for polishing the writing, drafting figures, brainstorming, and proposing keywords. We have reviewed all AI-assisted work. All AI-generated content, including experimental designs, code, figures, and text, was manually reviewed, verified, and revised as needed by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects or personally identifiable data, and all experiments use publicly available datasets and models in accordance with their licenses. We are not aware of any ethical concerns specific to this work that must be highlighted here.

REPRODUCIBILITY STATEMENT

To support reproducibility, we report the complete description of the methods, experimental settings and details, and hyperparameters, together with details on the datasets and evaluation protocols to reproduce the work, in Appendix B.

REFERENCES

Gaon An, Seungyong Moon, Jang-Hyun Kim, and Hyun Oh Song. Uncertainty-based offline reinforcement learning with diversified q-ensemble. In *Advances in Neural Information Processing Systems*, 2021.

Arpit Bansal, Hong-Min Chu, Avi Schwarzschild, Soumyadip Sengupta, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Universal guidance for diffusion models. In *International Conference on Learning Representations*, 2024.

Huayu Chen, Cheng Lu, Chengyang Ying, Hang Su, and Jun Zhu. Offline reinforcement learning via high-fidelity generative behavior modeling. In *International Conference on Learning Representations*, 2023.

Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *arXiv preprint arXiv:2105.05233*, 2021.

Zihan Ding and Chi Jin. Consistency models as a rich and efficient policy class for reinforcement learning. In *International Conference on Learning Representations*, 2024.

Carles Domingo-Enrich, Michal Drozdzal, Brian Karrer, and Ricky TQ Chen. Adjoint matching: Fine-tuning flow and diffusion generative models with memoryless stochastic optimal control. In *International Conference on Learning Representations*, 2025.

Perry Dong, Qiyang Li, Dorsa Sadigh, and Chelsea Finn. Expo: Stable reinforcement learning with expressive policies. In *International Conference on Learning Representations*, 2026a.

Yonghoon Dong, Kyungmin Lee, Changyeon Kim, Jaehyuk Kim, and Jinwoo Shin. Trust region Q adjoint matching. *arXiv preprint arXiv:2605.27079*, 2026b.

JaeHyeok Doo, Byeongguk Jeon, Seonghyeon Ye, Kimin Lee, and Minjoon Seo. Q-Flow: Stable and expressive reinforcement learning with flow-based policy. In *International Conference on Machine Learning*, 2026.

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. In *International Conference on Learning Representations*, 2025.

Ruiqi Feng, Chenglei Yu, Wenhao Deng, Peiyan Hu, and Tailin Wu. On the guidance of flow matching. In *International Conference on Machine Learning*, 2025.

Scott Fujimoto and Shixiang (Shane) Gu. A minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2021.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, 2018.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, 2019.

Seyed Kamyar Seyed Ghasemipour, Dale Schuurmans, and Shixiang Shane Gu. EMaQ: Expected-max q-learning operator for simple yet effective offline and online RL. In *International Conference on Machine Learning*, 2021.

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.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2020.

Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *IEEE International Conference on Robotics and Automation*, 2019.

Bingyi Kang, Xiao Ma, Chao Du, Tianyu Pang, and Shuicheng Yan. Efficient diffusion policies for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2021.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

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

Qiyang Li and Sergey Levine. Q-learning with adjoint matching. In *International Conference on Learning Representations*, 2026.

Qiyang Li, Zhiyuan Zhou, and Sergey Levine. Reinforcement learning with action chunking. In *Advances in Neural Information Processing Systems*, 2025.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations*, 2023.

Zhen Liu, Tim Z. Xiao, Carles Domingo-Enrich, Weiyang Liu, and Dinghuai Zhang. Value gradient guidance for flow matching alignment. In *Advances in Neural Information Processing Systems*, 2025.

Lei Lyu, Yunfei Li, Yu Luo, Fuchun Sun, Tao Kong, Jiafeng Xu, and Xiao Ma. Flow-based policy for online reinforcement learning. In *Advances in Neural Information Processing Systems*, 2025.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning*, 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

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

Seohong Park, Qiyang Li, and Sergey Levine. Flow q-learning. In *International Conference on Machine Learning*, 2025b.

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.

Michael Psenka, Alejandro Escontrela, Pieter Abbeel, and Yi Ma. Learning a diffusion model policy from rewards via q-score matching. In *International Conference on Machine Learning*, 2024.

Mahmoud Selim, Cristina Cipriani, and Karl H. Johansson. Noisy-space policy gradient for diffusion policies in offline reinforcement learning. In *International Conference on Machine Learning*, 2026.

Hikmet Simsir and Ozgur S. Oguz. Lagrangian perturbation diffusion steering: Latent reinforcement learning for generative policies. In *International Conference on Machine Learning*, 2026.

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*, 2021.

Entong Su, Tyler Westenbroek, Anusha Nagabandi, and Abhishek Gupta. Rfs: Reinforcement learning with residual flow steering for dexterous manipulation. In *International Conference on Learning Representations*, 2026.

Denis Tarasov, Vladislav Kurenkov, Alexander Nikulin, and Sergey Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

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. In *Conference on Robot Learning*, 2025.

Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. In *International Conference on Learning Representations*, 2023.

Ziqian Wang, Rui Zhang, Yitian Liu, Xingjian Mao, Minqian Wang, and Yao Mu. Q-vgm: Q-guided value-gradient matching for offline-to-online rl of flow-matching vla policies. *arXiv preprint arXiv:2606.08015*, 2026.

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

Zhiyuan Zhou, Andy Peng, Charles Xu, Qiyang Li, Tobias Springenberg, Kevin Frans, and Sergey Levine. Test-time gradient guidance of flow policies in reinforcement learning. *arXiv preprint arXiv:2606.11087*, 2026.

A GUIDANCE SIGNAL DETAILS

This section details the three guidance signals used in controlled analysis.

A.1 NOISY-SPACE CRITIC

Noisy-space critic learning associates an intermediate action with the value of its endpoint under the BC flow (Doo et al., 2026). For training, we sample $(s, a) \sim \mathcal{D}$, $a_0 \sim \mathcal{N}(0, I_{d_a})$, and $t \sim \mathcal{U}[0, 1]$, and construct $a_t = (1-t)a_0 + ta$. We then follow the BC flow from a_t to obtain $a_1 = \mathrm{ODE}_{t\to 1}(s, a_t)$, where $\mathrm{ODE}_{t\to 1}$ denotes the remaining rollout under v_{BC}. This endpoint need not equal the dataset action a used to construct the interpolant.

The time-conditioned critic Q_ω is trained to predict the endpoint value supplied by the target action-space critic:

$$\mathcal{L}_{\mathrm{NSC}}(\omega) = \mathbb{E}_{(s,a)\sim\mathcal{D},\ a_0\sim\mathcal{N}(0,I_{d_a}),\ t\sim\mathcal{U}[0,1]}\left[\left(Q_\omega(s, a_t, t) - Q_{\bar{\phi}}\big(s, \mathrm{ODE}_{t\to 1}(s, a_t)\big)\right)^2\right]. \quad (9)$$

Only ω is optimized in this objective; the BC rollout and target critic supply fixed regression targets. The guidance signal is

$$g(s, a_t, t) = \nabla_{a_t} Q_\omega(s, a_t, t).$$

A.2 ONE-STEP REPARAMETERIZED GUIDANCE

The remaining BC rollout is approximated by one Euler step:

$$\hat{a}_1 = a_t + (1-t)v_{\mathrm{BC}}(s, a_t, t).$$

Differentiating the predicted endpoint value with respect to a_t gives

$$g(s, a_t, t) = \nabla_{a_t} Q_\phi(s, \hat{a}_1) = \left[I_{d_a} + (1-t)\frac{\partial v_{\mathrm{BC}}(s, a_t, t)}{\partial a_t}\right]^\top \nabla_{\hat{a}_1} Q_\phi(s, \hat{a}_1).$$

This construction differentiates through the one-step prediction rather than the full BC rollout. The BC parameters remain unchanged.

A.3 ONE-STEP QGF

QGF uses the same endpoint prediction but approximates its Jacobian by the identity (Zhou et al., 2026):

$$\frac{\partial \hat{a}_1}{\partial a_t} = I_{d_a} + (1-t)\frac{\partial v_{\mathrm{BC}}(s, a_t, t)}{\partial a_t} \approx I_{d_a}.$$

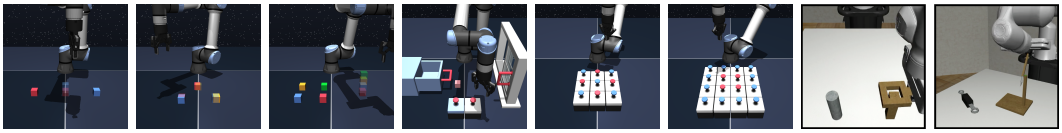
The resulting guidance signal is

$$g(s, a_t, t) = \nabla_{\hat{a}_1} Q_\phi(s, \hat{a}_1).$$

This gradient is applied directly to a_t, without backpropagating through the BC velocity. Both one-step constructions recompute the endpoint prediction after each guidance update.

Table 3: **Environment and dataset details.** For ROBOMIMIC, the dataset type indicates the human demonstrations used to train the data-collection policy. d_a denotes the dimension of the full action chunk, i.e., the single-step action dimension (5 for OGBENCH, 7 for ROBOMIMIC) multiplied by the chunk size.

Benchmark	Environment	Default task	Dataset type	Dataset size	Chunk size	d_a	Horizon
OGBENCH	`cube-double`	`task2`	`play`	1M	5	25	500
	`cube-triple`	`task2`	`play`	10M	5	25	1000
	`cube-quadruple`	`task2`	`play`	100M	5	25	1000
	`scene`	`task2`	`play`	1M	5	25	750
	`puzzle-3x3`	`task4`	`play`	1M	5	25	500
	`puzzle-4x4`	`task4`	`play`	10M	5	25	500
ROBOMIMIC	`square`	–	`mh`	100K	4	28	500
	`tool-hang`	–	`ph`	100K	4	28	500



`cube-double` `cube-triple` `cube-quadruple` `scene` `puzzle-3x3` `puzzle-4x4` `square` `tool-hang`

Figure 8: **Environments.** The first six are OGBENCH manipulation environments; the last two are ROBOMIMIC tasks.

B EXPERIMENTAL DETAILS

B.1 ENVIRONMENTS AND TASKS

We evaluate on robotic manipulation tasks from two benchmarks, OGBENCH and ROBOMIMIC, shown in Figure 8.

OGBENCH (Park et al., 2025a). OGBENCH is a benchmark originally designed for offline goal-conditioned RL, whose manipulation environments share a common setup in which a robot arm interacts with objects on a table. We use six environments:

- `cube-double`, `cube-triple`, and `cube-quadruple`: rearrange two, three, or four cubes into a target configuration.
- `scene`: manipulate a cube, a drawer, a window, and lock buttons, completing several subtasks in the right order.
- `puzzle-3x3` and `puzzle-4x4`: press buttons on a grid, where each press toggles the button and its neighbors, to reach a target pattern.

To benchmark reward-maximizing offline RL, we use the single-task variants of these environments, following Park et al. (2025b). Each environment defines five different goals, each specifying a different task (`singletask-task1` to `singletask-task5`). Throughout the analysis in Section 4, Section 5.2, and sparse guidance experiments in Section 5.3, we use default task provided in each environment, and for comparison to baselines in Table 2, we use all five tasks in each environment. An episode ends once the task's goal is achieved, and we report the binary task success rate.

ROBOMIMIC (Mandlekar et al., 2021). ROBOMIMIC is a benchmark of robotic manipulation tasks with human teleoperation data, primarily used to evaluate imitation learning algorithms, as well as to provide demonstrations for online RL. In this work, we instead use it for offline RL on two tasks:

- `square`: pick up a square nut and place it onto a peg.
- `tool-hang`: assemble a frame from a hook and a base, then hang a tool on it.

Algorithm 1 Localized guidance (Section 4.1)

Require: State s, BC velocity field v_{BC}, number of flow steps N, guidance timestep $t_{\mathrm{g}} \in \{0, \frac{1}{N}, \dots, \frac{N-1}{N}\}$, guidance steps K, guidance coefficient α, guidance signal g

Sample $a_0 \sim \mathcal{N}(0, I_{d_a})$
for $t = 0, \frac{1}{N}, \dots, \frac{N-1}{N}$ **do**
  if $t = t_{\mathrm{g}}$ **then**
    $a_t^{(0)} \leftarrow a_t$
    for $k = 0, \dots, K-1$ **do**
      $a_t^{(k+1)} \leftarrow a_t^{(k)} + \alpha\, g(s, a_t^{(k)}, t)$ ▷ Eq. (6)
    end for
    $a_t \leftarrow a_t^{(K)}$
  end if
  $a_{t+1/N} \leftarrow a_t + \frac{1}{N}\, v_{\mathrm{BC}}(s, a_t, t)$ ▷ Eq. (3)
end for
return a_1

Algorithm 2 Dense guidance

Require: State s, BC velocity field v_{BC}, number of flow steps N, guidance coefficient α, guidance signal g

Sample $a_0 \sim \mathcal{N}(0, I_{d_a})$
for $t = 0, \frac{1}{N}, \dots, \frac{N-1}{N}$ **do**
  $a_{t+1/N} \leftarrow a_t + \frac{1}{N}\big[\, v_{\mathrm{BC}}(s, a_t, t) + \alpha\, g(s, a_t, t) \,\big]$ ▷ Eq. (4)
end for
return a_1

As in OGBENCH, we report the binary task success rate.

B.2 DATASETS AND REWARD DEFINITION

The two benchmarks represent different data regimes for offline RL: OGBENCH datasets contain diverse behavior not directed at any single task, whereas offline RL datasets we generated on ROBOMIMIC are collected by a BC policy trained on demonstrations. This contrast determines how capable the BC policy is on its own, and thus how much room is left for value guidance to improve upon it, which is central to several findings in the main text. Table 3 summarizes the datasets.

OGBENCH. Each environment provides a `play` dataset, which is collected by a scripted policy performing random, noisy behaviors rather than solving any particular task, so only a small fraction of it resembles a solution to the evaluated task. We use the standard 1M-transition datasets for `cube-double`, `scene`, and `puzzle-3x3`, and larger datasets for the harder environments: 10M transitions for `cube-triple` and `puzzle-4x4`, and 100M transitions for `cube-quadruple`.

ROBOMIMIC. Since ROBOMIMIC's human demonstrations are almost entirely successful, we construct our own offline datasets from them. We train a BC flow policy for 3K epochs on the multi-human (`mh`) demonstrations for `square` and the proficient-human (`ph`) demonstrations for `tool-hang`, and roll it out to collect 100K transitions per task, rewarding only the transition that achieves task success in each rollout. This same policy also serves as the pretrained BC flow policy in all ROBOMIMIC experiments. The resulting datasets contain both successful and failed attempts, with the BC policy succeeding in about 50% of episodes on `square` and 30% on `tool-hang`.

B.3 GUIDANCE SAMPLING PROCEDURES

Algorithms 1 and 2 describe action sampling procedure with localized and dense guidance, respectively. In both, g denotes any of the guidance signals in Table 1. Under guided bootstrapping in Section 5.1, $\alpha\, g$ is replaced by the amortized guidance field e_ψ.

B.4 Controlled Analysis Details (Section 4)

Policy and critic training. For each task and seed, we first train a BC flow policy and then train the critic with EMaQ (Eq. (5)), using $M = 8$ candidate actions on OGBench and $M = 4$ on Robomimic, with training steps listed in Table 5. Each seed thus has its own BC policy and critic, which are kept fixed afterward and shared by every guidance configuration within that seed and (t, K, B) configuration. For all three guidance signals listed in Table 1, the guidance gradient is computed from the online critic, aggregating the critic ensemble as reported in Table 5.

Guidance sweep and evaluation. We sweep all three guidance signals over flow timesteps $t \in \{0, 0.4, 0.6, 0.8\}$ and guidance steps $K \in \{1, 3, 5, 10\}$, with budgets $B \in \{0.05, 0.1, 0.2, 0.3, 0.5\}$ on OGBench and $B \in \{0.01, 0.05, 0.1, 0.2\}$ on Robomimic. For every evaluated configuration of guidance signal, t, K, and B, the guidance coefficient α is calibrated separately so that the guidance magnitude in Eq. (7) matches B, and then held fixed during evaluation. Each configuration is evaluated with 50 episodes per seed over four seeds, for a total of 200 evaluation episodes per cell of the guidance scaling matrices. Figure 1 reports, for each (t, K), the budget with the highest mean success across seeds, and Figure 2 further selects the best guidance timestep and budget for each K, averaged over the tasks of each benchmark. We provide the full results over every budget without selection in Figures 10 and 11.

Calibration details. For each configuration of flow timestep **(T)**, iterations **(I)**, and guidance signal **(S)**, we calibrate α so that the guidance budget in Eq. (7) matches the target value B. We estimate Eq. (7) on 1,024 states sampled from the dataset, each paired with eight noise samples $a_0 \sim \mathcal{N}(0, I_{d_a})$, by running the BC flow up to t and applying the K guidance updates of Eq. (6). We then adjust α until the estimate matches the target budget value. Figure 9 shows the resulting calibrated values: the same budget requires different α across tasks and signals, with the calibrated α approximately inversely proportional to the normalized gradient norm $\|g\|/\sqrt{d_a}$. Specifying α alone would therefore apply very different amounts of guidance across configurations, which the guidance budget introduced in this work would avoid.

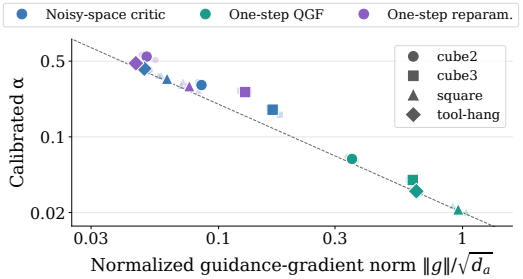


Figure 9: α **calibration.** The same budget requires different α across tasks, following differences in gradient magnitude.

B.5 Guided Bootstrapping Analysis Details (Section 5.2)

Policy and critic training. All configurations start from the same pretrained BC flow policy, which is kept fixed, and train the critic and guidance field jointly for the number of offline RL steps listed in Table 5. The critic is trained with the TD objective in Eq. (1), where each bootstrap action is generated by the current guidance configuration as described in Section 5.1, and the guidance field is simultaneously regressed onto the scaled guidance signal of the current critic as in Eq. (8). For noisy-space guidance, the noisy-space critic is updated alongside as in Eq. (9), with its regression targets computed from the target critic, so that the guidance signal tracks the critic being learned.

Maintaining the guidance budget. Following Dong et al. (2026b) and Simsir & Oguz (2026), we adapt α online so that the realized guidance magnitude tracks the target budget B. At each training step we estimate $\widehat{B}$ from the bootstrap actions in the minibatch, as the cumulative norm of the guidance over the K guidance steps normalized by $\sqrt{d_a}$, following Eq. (7). To reduce variance we smooth this estimate with an exponential moving average,

$$\bar{B} \leftarrow (1 - \rho)\, \bar{B} + \rho\, \widehat{B}, \tag{10}$$

where $\widehat{B}$ is the per-step estimate and $\rho \in (0, 1]$ is the smoothing coefficient, so that $\bar{B}$ tracks the realized magnitude over recent steps rather than a single minibatch sampled at each training step. We then adapt α by projected dual descent:

$$\alpha_{n+1} \leftarrow \Pi_{\mathcal{A}}\big(\alpha_n + \eta_\alpha (B - \bar{B})\big), \tag{11}$$

Table 4: **Guidance schedules used in Section 5.3.**

Schedule	Benchmark	Guidance timesteps t	Steps per timestep	Total guidance steps
Dense	Both	$\{0, 0.1, \ldots, 0.9\}$	1	10
Window-3	Both	$\{0.7, 0.8, 0.9\}$	1	3
Window-5	Both	$\{0.5, 0.6, 0.7, 0.8, 0.9\}$	1	5
Localized	OGBENCH	$\{0.8\}$	3	3
		$\{0.6\}$	5	5
Localized	ROBOMIMIC	$\{0.4\}$	1	1
		$\{0.4\}$	3	3
		$\{0.4\}$	5	5

where α_n is the coefficient after n updates, $\eta_\alpha > 0$ is the dual step size, and $\Pi_{\mathcal{A}}$ projects onto a bounded range $\mathcal{A}$. When the realized magnitude falls below the budget, $B - \bar{B} > 0$ and α rises, making the guidance more aggressive; when it exceeds the budget the step is negative and α decreases. Since α scales the regression target of the guidance field, this adjusts the magnitude of the guidance the field learns to produce. In all experiments we use $\alpha_0 = 0.05$, $\eta_\alpha = 0.01$, $\rho = 0.1$, and $\mathcal{A} = [0, 20]$.

Guidance sweep and evaluation. We train noisy-space guidance and one-step QGF over flow timesteps $t \in \{0, 0.4, 0.6, 0.8\}$ and guidance steps $K \in \{1, 3, 5, 10\}$ with budgets $B \in \{0.1, 0.2, 0.3\}$ on OGBENCH, and over $t \in \{0, 0.4, 0.8\}$ and $K \in \{1, 3, 5\}$ with $B \in \{0.01, 0.05, 0.1\}$ on ROBOMIMIC. Each configuration is trained with four seeds and evaluated at the final training step with 50 episodes per seed, totaling in 200 episodes per (t, K, B) configuration. As in the controlled analysis, Figure 4 reports, for each (t, K), the budget with the highest performance, and Figure 12 reports every budget without selection. Figure 5 further selects the best flow timestep and budget for each K and averages over the tasks of each benchmark, where the dense guidance results use the configurations of Table 2 on the same tasks.

B.6 BASELINE COMPARISON DETAILS (TABLE 2)

Unlike the preceding analyses, which use only the default task, this comparison evaluates all five tasks of each OGBENCH environment along with ROBOMIMIC tasks. All methods use the common hyperparameters in Table 5, and each is evaluated at the final training step with 50 episodes per seed over four seeds. Detailed explanations of the baselines are provided in Appendix C.

OGBENCH tuning. For the baselines, we adopt the per-environment hyperparameters reported as optimal in previous works (Li & Levine, 2026; Dong et al., 2026b). For noisy-space guidance and one-step QGF with guided bootstrapping, we tune the guidance budget B on the default task of each environment, sweeping over $B \in \{0.1, 0.2, 0.3\}$ and use the selected budget for all five tasks of the corresponding environment.

ROBOMIMIC tuning. In ROBOMIMIC, we sweep the task-specific hyperparameter of each method over the ranges in Table 7 and selected the value with the highest final success rate averaged over seeds. Here we manually sweep the main hyperparameter of each method, including ours, over the ranges in Table 7 with four seeds per configuration, and select the task-specific hyperparameter with the highest success rate. The selected values are listed in Table 6.

B.7 SPARSE GUIDANCE DETAILS (SECTION 5.3)

We compare dense guidance with two kinds of sparse schedules, listed in Table 4, each of which applies either three or five guidance steps in total. Window schedules apply one guidance step at each of several consecutive late flow timesteps, which we choose without any per-task search, since later timesteps are effective for both guidance signals in Sections 4.2 and 5.2. Localized schedules

instead apply all guidance steps at a single timestep, which we select from the guided bootstrapping results in Figure 4. On OGBENCH, we select $t = 0.8$ for $K = 3$ and $t = 0.6$ for $K = 5$ from the default tasks of `cube-double`, `cube-triple`, and `puzzle-4x4`, and apply the same schedules unchanged to `scene`, `puzzle-3x3`, and `cube-quadruple`; we omit $K = 1$, since a single guidance step is insufficient on OGBENCH. On ROBOMIMIC, we select $t = 0.4$ for all $K \in \{1, 3, 5\}$.

Each schedule is trained with guided bootstrapping using the same schedule to generate bootstrap actions, with the budget sweep of $B \in \{0.1, 0.2, 0.3\}$, and is evaluated on the default tasks of the six OGBENCH environments and on both ROBOMIMIC tasks with 50 episodes per seed over four seeds. We measure latency as the time to generate one action chunk on ROBOMIMIC, including both the BC integration steps and the guidance steps.

Table 5: **Common hyperparameters.** Values that differ across tasks within a benchmark are listed per task.

(a) OGBENCH

Hyperparameter	Value
Batch size	256
Optimizer	Adam (Kingma & Ba, 2015)
Learning rate	3×10^{-4}
Target network update rate	0.005
Discount factor γ	0.995 (`cube-{}`), 0.999 (`scene, puzzle-{}`)
Critic ensemble size	2
Critic aggregation	mean
UTD ratio	1
Number of flow steps	10
BC pretraining	300K gradient steps
Offline RL training steps	1M
Network width	512 (1M data), 1024 (10M/100M data)
Network depth	4
Actor layer norm	True
Critic layer norm	True

(b) ROBOMIMIC

Hyperparameter	Value
Batch size	256
Optimizer	Adam
Learning rate	3×10^{-4}
Target network update rate	0.005
Discount factor γ	0.99
Critic ensemble size	2
Critic aggregation	min
UTD ratio	1
Number of flow steps	10
BC pretraining	3K epochs
Offline RL training steps	200K
Network width	512
Network depth	4
Actor layer norm	False
Critic layer norm	True

Table 6: **Task-specific hyperparameters.**

Benchmark	Environment	QAM τ	TRQAM ε_{KL}	IFQL κ	DSRL σ	FEdit λ	NSC + dense B	QGF + dense B
OGBENCH	`cube-double`	1.0	0.5	0.9	1.0	0.2	0.2	0.1
	`cube-triple`	3.0	0.5	0.95	1.4	0.3	0.1	0.1
	`cube-quadruple`	1.0	1.0	0.95	1.4	0.4	0.2	0.2
	`scene`	1.0	0.5	0.9	0.4	0.2	0.2	0.2
	`puzzle-3x3`	3.0	2.0	0.95	1.0	0.2	0.2	0.2
	`puzzle-4x4`	30.0	4.0	0.9	1.0	0.8	0.2	0.2
ROBOMIMIC	`square`	0.01	0.1	0.7	1.0	0.01	0.05	0.1
	`tool-hang`	0.01	0.01	0.7	0.6	0.01	0.05	0.05

Table 7: **Hyperparameter sweep ranges on ROBOMIMIC.**

Method	Hyperparameter	Sweep range
`QAM`	τ	$\{0.01, 0.03, 0.1, 0.3, 1.0\}$
`TRQAM`	ε_{KL}	$\{0.01, 0.05, 0.1, 0.5, 1.0\}$
`IFQL`	κ	$\{0.5, 0.6, 0.7, 0.8, 0.9\}$
`DSRL`	σ	$\{0.2, 0.4, 0.6, 0.8, 1.0, 1.2\}$
`FEdit`	λ	$\{0.01, 0.05, 0.1, 0.2, 0.4\}$
`NSC + dense`	B	$\{0.01, 0.05, 0.1\}$
`QGF + dense`	B	$\{0.01, 0.05, 0.1\}$

C Baseline Explanations

QAM (Li & Levine, 2026). QAM trains a velocity field v_θ through adjoint matching, using v_{BC} as the behavioral reference. Its critic bootstraps from the resulting flow policy π_θ:

$$\mathcal{L}_Q(\phi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D},\ a'\sim\pi_\theta(\cdot|s')}\left[\left(Q_\phi(s,a) - r - \gamma Q_{\bar{\phi}}(s',a')\right)^2\right]. \tag{12}$$

Here π_θ generates actions by following v_θ from Gaussian noise. For policy optimization, QAM instead samples stochastic trajectories using

$$da_t = \left[2v_\theta(s,a_t,t) - \frac{a_t}{t}\right]dt + \sigma(t)\,dW_t, \qquad \sigma(t) = \sqrt{\frac{2(1-t)}{t}},$$

where W_t is Brownian motion. Starting from the terminal condition $p_1 = -\tau\nabla_{a_1}Q_\phi(s,a_1)$, the lean adjoint is computed backward along each trajectory:

$$dp_t = -\left[\frac{\partial}{\partial a_t}\left(2v_{\mathrm{BC}}(s,a_t,t) - \frac{a_t}{t}\right)\right]^\top p_t\,dt.$$

The coefficient τ controls the critic contribution. The policy minimizes

$$\mathcal{L}_{\mathrm{QAM}}(\theta) = \mathbb{E}_{s,\{a_t\}}\left[\int_0^1 \left\| \frac{2\big(v_\theta(s,a_t,t) - v_{\mathrm{BC}}(s,a_t,t)\big)}{\sigma(t)} + \sigma(t)p_t \right\|_2^2 dt\right].$$

Sampled trajectories and adjoints are held fixed during this update, avoiding backpropagation through the sampling chain.

TRQAM (Dong et al., 2026b). TRQAM retains the critic objective in Eq. (12) and adapts adjoint matching to control deviation from the behavioral reference. It introduces $\lambda > 0$ through

$$\sigma_\lambda(t) = \frac{\sigma(t)}{\sqrt{\lambda}}, \qquad \sqrt{\lambda}\,\sigma_\lambda(t) = \sigma(t),$$

and minimizes

$$\mathcal{L}_{\mathrm{TRQAM}}(\theta) = \mathbb{E}_{s,\{a_t\}}\left[\int_0^1 \left\| \frac{2\big(v_\theta(s,a_t,t) - v_{\mathrm{BC}}(s,a_t,t)\big)}{\sigma_\lambda(t)} + \sigma_\lambda(t)p_t \right\|_2^2 dt\right].$$

The adjoint uses the terminal condition $p_1 = -\nabla_{a_1}Q_\phi(s,a_1)$. The total diffusion amplitude remains $\sigma(t)$; λ changes the balance between velocity deviation and the adjoint target.

For sampler step size h, the path-space KL is estimated by

$$\widehat{D}_{\mathrm{KL}} = \mathbb{E}_{s,\{a_t\}}\left[\sum_t \frac{2h}{\sigma(t)^2}\left\|v_\theta(s,a_t,t) - v_{\mathrm{BC}}(s,a_t,t)\right\|_2^2\right].$$

TRQAM smooths this estimate and applies projected dual descent:

$$\overline{D}_{\mathrm{KL}} \leftarrow (1-\rho)\overline{D}_{\mathrm{KL}} + \rho\widehat{D}_{\mathrm{KL}}, \qquad \lambda \leftarrow \max\left\{0, \lambda + \eta_\lambda\left(\overline{D}_{\mathrm{KL}} - \varepsilon_{\mathrm{KL}}\right)\right\}.$$

Thus, exceeding the prescribed KL bound increases λ, strengthening the relative penalty on deviation.

IFQL (Hansen-Estruch et al., 2023). IFQL is the flow-policy counterpart of IDQL, which combines in-sample value learning with sampling-based policy extraction from a BC policy. The critic Q_ϕ and a state-value function V_ξ are trained with implicit Q-learning (Kostrikov et al., 2021), which never queries actions outside the dataset:

$$\mathcal{L}(\phi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q_\phi(s,a) - r - \gamma V_\xi(s')\right)^2\right], \tag{13}$$

$$\mathcal{L}(\xi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[L_2^\kappa\left(Q_{\bar{\phi}}(s,a) - V_\xi(s)\right)\right], \tag{14}$$

where $L_2^\kappa(u) = |\kappa - \mathbb{1}(u<0)|\,u^2$ is the expectile loss, and the expectile $\kappa \in (0.5, 1)$ controls how closely V_ξ approximates the maximum of Q_ϕ over in-distribution actions. At evaluation, IFQL samples N candidate actions from the BC flow policy π_{BC} and executes the one with the highest value:

$$a^\star = \arg\max_{a\in\{a_{(1)},\ldots,a_{(N)}\}} Q_\phi(s,a), \qquad a_{(i)} \sim \pi_{\mathrm{BC}}(\cdot \mid s). \tag{15}$$

We use $N = 32$ candidates and tune κ per task (Table 6).

DSRL (Wagenmaker et al., 2025). DSRL learns a Gaussian noise policy $\pi_\psi(a_0 \mid s)$ whose output is passed through π_{BC}, producing $a_1 = \mathrm{ODE}_{0\to1}(s, a_0)$. The sampled noise is squashed by $\tanh$ and scaled by the noise scale σ, so that $a_0 \in [-\sigma, \sigma]^{d_a}$; σ thus bounds how far the noise policy can move away from the Gaussian prior of the BC flow, and is the main hyperparameter of DSRL. Given the action-space critic trained with TD bootstrapping in Eq. (1), DSRL additionally learns a noise-space critic which distills the value of the BC endpoint:

$$\mathcal{L}_{\mathrm{noise}}(\omega) = \mathbb{E}_{s\sim\mathcal{D},\ a_0\sim\mathcal{N}(0,I_{d_a})}\left[\left(Q_\omega(s,a_0) - Q_\phi\big(s, \mathrm{ODE}_{0\to1}(s,a_0)\big)\right)^2\right].$$

This corresponds to the noisy-space critic considered in this study, restricted to the initial noise at $t = 0$. In contrast to gradient-based value guidance methods, DSRL learns the noise policy by minimizing the entropy-regularized objective

$$\mathcal{L}_\pi(\psi) = \mathbb{E}_{s\sim\mathcal{D},\ a_0\sim\pi_\psi(\cdot|s)}\left[\tau \log \pi_\psi(a_0 \mid s) - Q_\omega(s,a_0)\right],$$

where τ is the entropy coefficient.

FEdit. FEdit adapts the action-editing component of EXPO (Dong et al., 2026a) to a BC flow. It samples $a_1 \sim \pi_{\mathrm{BC}}(\cdot \mid s)$ and $\Delta a \sim \pi_\psi(\cdot \mid s, a_1)$, then executes $\mathrm{clip}_{[-1,1]}(a_1 + \lambda\Delta a)$. Its critic objective is

$$\mathcal{L}_Q(\phi) = \mathbb{E}_{\substack{(s,a,r,s')\sim\mathcal{D} \\ a_1'\sim\pi_{\mathrm{BC}}(\cdot|s'),\ \Delta a'\sim\pi_\psi(\cdot|s',a_1')}}\left[\left(Q_\phi(s,a) - r - \gamma Q_{\bar\phi}\big(s', \mathrm{clip}_{[-1,1]}(a_1' + \lambda\Delta a')\big)\right)^2\right].$$

The Gaussian edit policy minimizes

$$\mathcal{L}_\pi(\psi) = \mathbb{E}_{\substack{s\sim\mathcal{D},\ a_1\sim\pi_{\mathrm{BC}}(\cdot|s) \\ \Delta a\sim\pi_\psi(\cdot|s,a_1)}}\left[\tau \log \pi_\psi(\Delta a \mid s, a_1) - Q_\phi\big(s, \mathrm{clip}_{[-1,1]}(a_1 + \lambda\Delta a)\big)\right].$$

Here λ controls the edit scale. FEdit uses one edited action for bootstrapping and evaluation, without EXPO's BoN sampling.

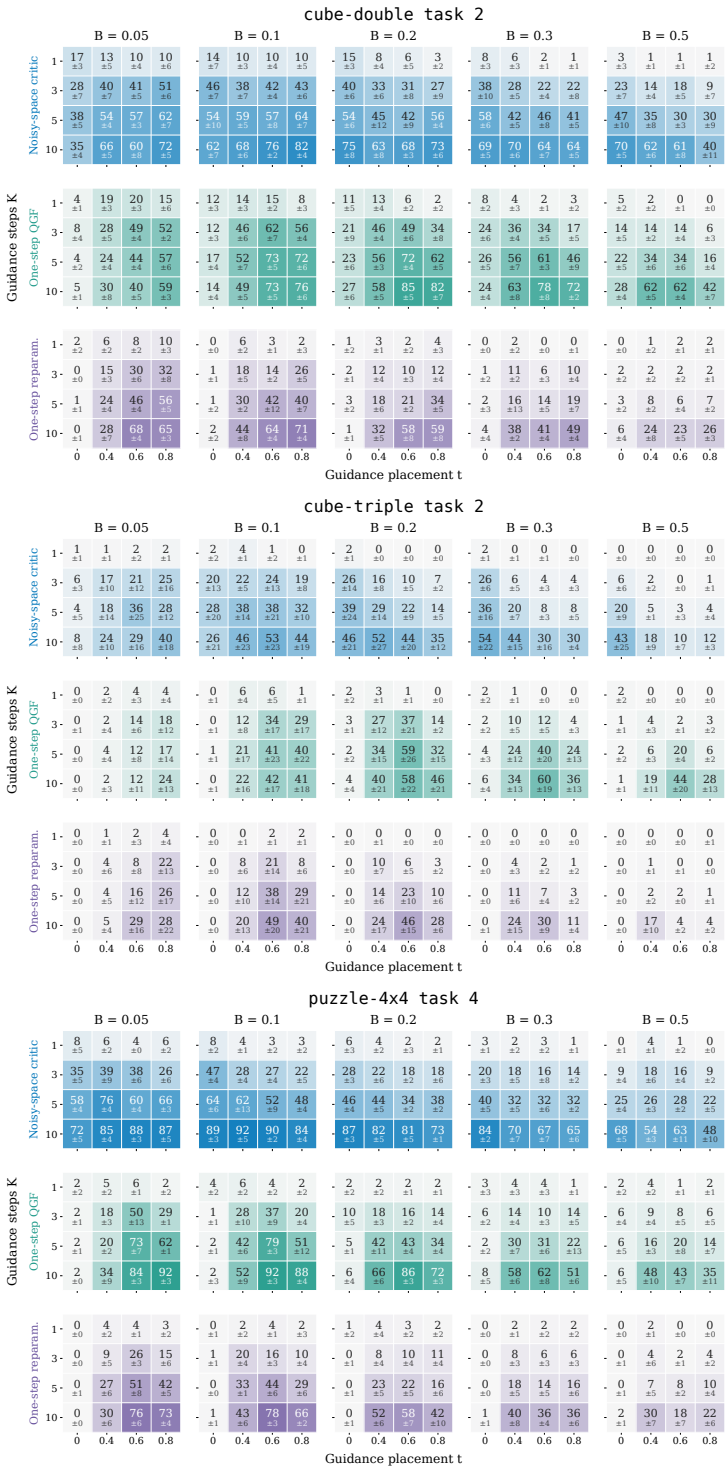


Figure 10: **Controlled guidance results on OGBench.**

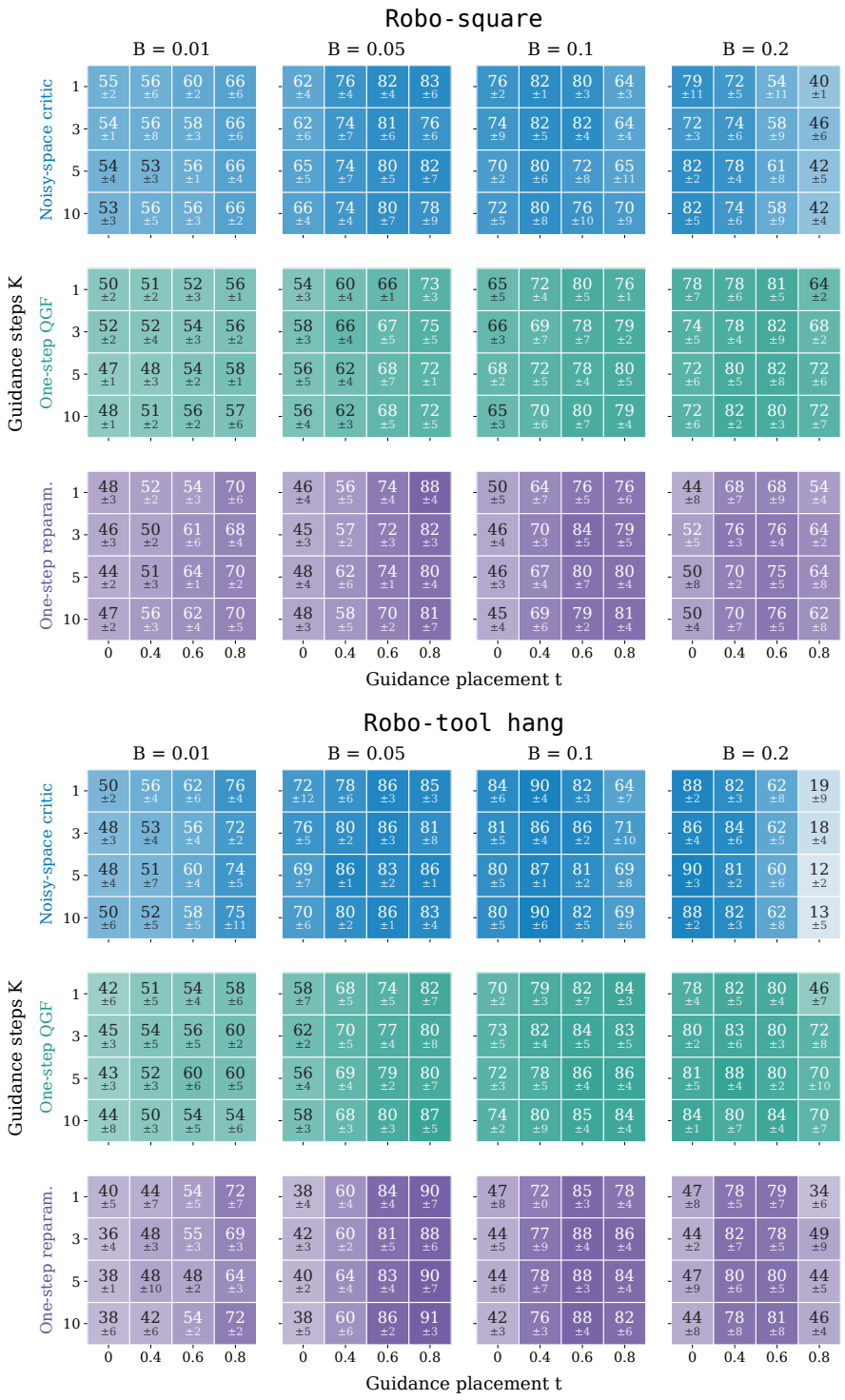


Figure 11: **Controlled guidance results on ROBOMIMIC.**

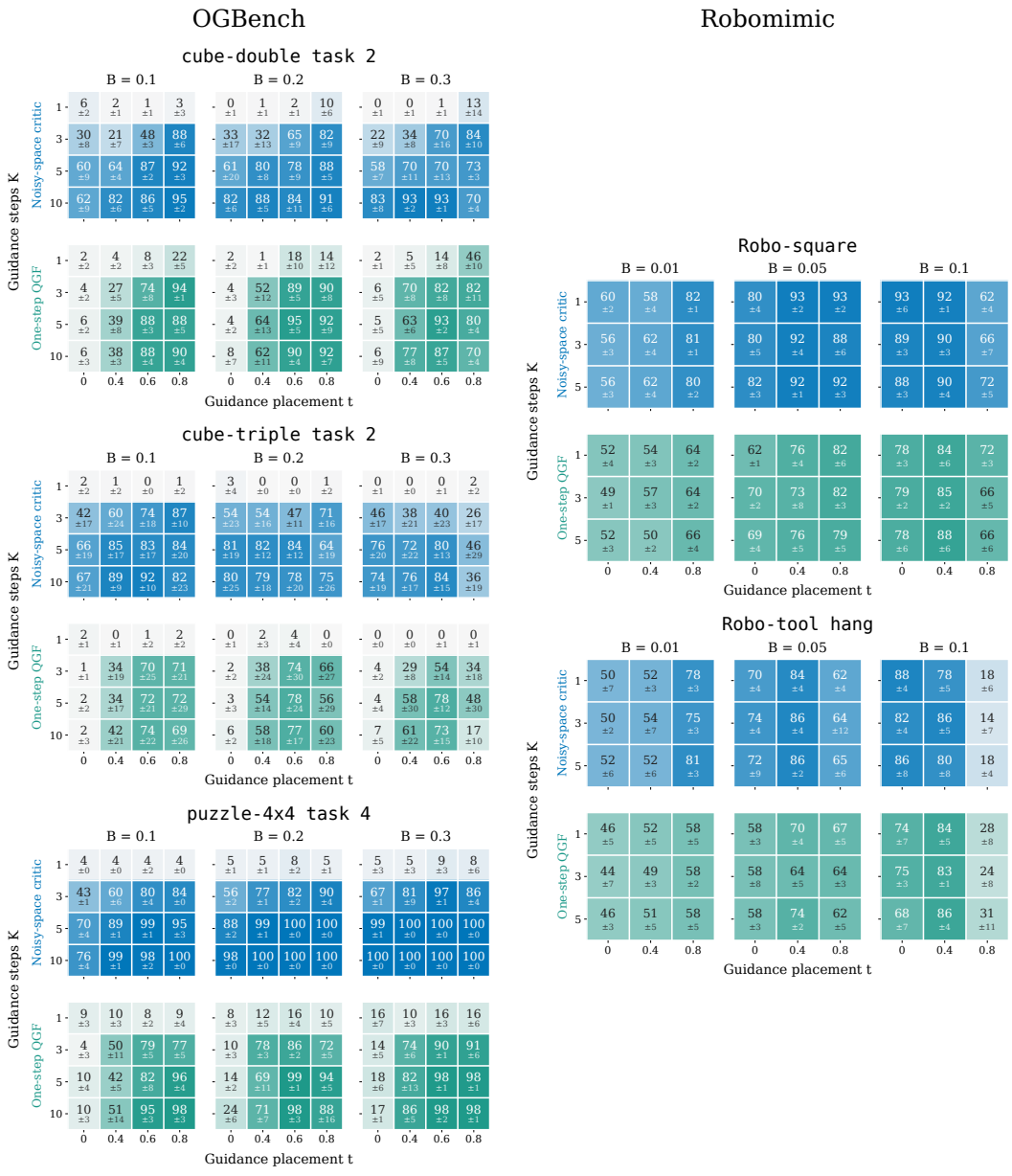


Figure 12: **Training-time results on OGBENCH (left) and ROBOMIMIC (right).**