Modular Diffusion Policy Training: Decoupling and Recombining Guidance and Diffusion for Offline RL

Anonymous authors
Paper under double-blind review

Abstract

In classifier-free diffusion(CFD), the diffusion model and its guidance are typically learned jointly and applied jointly in the inference stage. Before the guidance has converged, it provides unstable or even misleading gradients, which leads to inefficiency and instability during the early stage of training. Such strict coupling not only leads to self-enforcing variance and biased errors but also prevents the guidance module from being reused across different diffusion models. We propose Guidance-First Diffusion Training (GFDT), which pretrains and freezes the guidance model before diffusion policy learning. GFDT reduces peak memory and computation by 38.1%, decreases diffusion training by 65.6% and 27.66%, and achieves up to 43.16% and 60.98% performance improvements on offline RL benchmarks. Beyond efficiency, we uncover a strong plug-and-play property: replacing the guidance module only at inference time can substantially improve stability. Cross-algorithm swaps (e.g., Implicit Q-Learning (IDQL) guidance for Diffusion Q-Learning (DQL) policies) perform comparably to the stronger of the two, despite never being co-trained. Our theoretical analysis shows that GFDT enables the convergence on an optimal guidance and theoretically proves that it speeds up the training. Also, we proved that plug-and-play remains valid as long as the guidance and the diffusion model are trained with the same data distribution. Limitations arising from dataset mismatch are analyzed in detail, which further underscores the necessity of distributional alignment. This work opens a new line of research by treating diffusion and guidance as modular units that can be recombined, rather than as a monolithic process, suggesting a paradigm that may guide the future development of diffusion-based reinforcement learning.

1 Introduction

By formulating policy learning as a conditional generative process, **diffusion policies** are capable of modeling complex, multimodal behaviors and have demonstrated strong empirical performance across a wide range of RL and robotic manipulation tasks. A central challenge in diffusion policy learning lies in injecting reward optimization into the denoising process, especially in reinforcement learning, where reward is the main evaluation criterion. One of the most widely used methods is **Classifier-Free Guidance (CFG)**[1], which biases sampling toward high-reward trajectories.

Currently, the guidance module and the diffusion model are trained jointly and tightly coupled in inference. The TWO CHALLENGES that exist are: 1)The role of the guidance module is to steer the diffusion process toward generating actions that lead to higher rewards. However, before the guidance module has converged, it cannot provide effective guidance. Therefore, during the early stages of training, the diffusion model receives little to no meaningful directing from the guidance module, which misguides the diffusion process and degrades its performance Kim et al. (2023). 2) This tightly bound diffusion and guidance model can have common idiosyncratic errors and cause instability(large variance); the coupled usage prevents reuse across different combinations of network modules.

[1]All abbreviations are summarized in the Appendix.J

Existing methods solve Challenge 1 by pretraining the diffusion model and then letting the diffusion model generate samples for the guidance training. However, since the pre-trained diffusion model can only generate in-distribution samples without real environment interaction, the additional data it produces introduces limited new information in offline settings. In this work, we take the opposite approach: pre-training the guidance module, then freezing it to guide the diffusion model, which is called Guidance-First Diffusion Training (GFDT). This design is safe, since the guidance module is trained using observations, actions, and rewards from the dataset; the data is not mixed with the data generated by the diffusion model. This separation does not reduce exploration, as offline reinforcement learning relies entirely on fixed datasets. Moreover, it is beneficial, as the pretrained guidance module can speed up diffusion training while reducing computational cost.

To solve Challenge 2, CFG methods can be modularized: Further, we cross-combined components from models trained under different frameworks. Specifically, we took the guidance module from an IDQL model and paired it with a completely separate DQL diffusion model—one that had never been trained in conjunction with the IDQL guidance. The plug-and-play (PAP) module and its reversed model are functional, and the performance is comparable with the stronger of these two, and has a significantly better early training stage, outperforming both the baselines. This result strongly suggests that the relationship between the guidance module and the diffusion model is modular and flexible, and therefore, highlights a promising direction for future research.

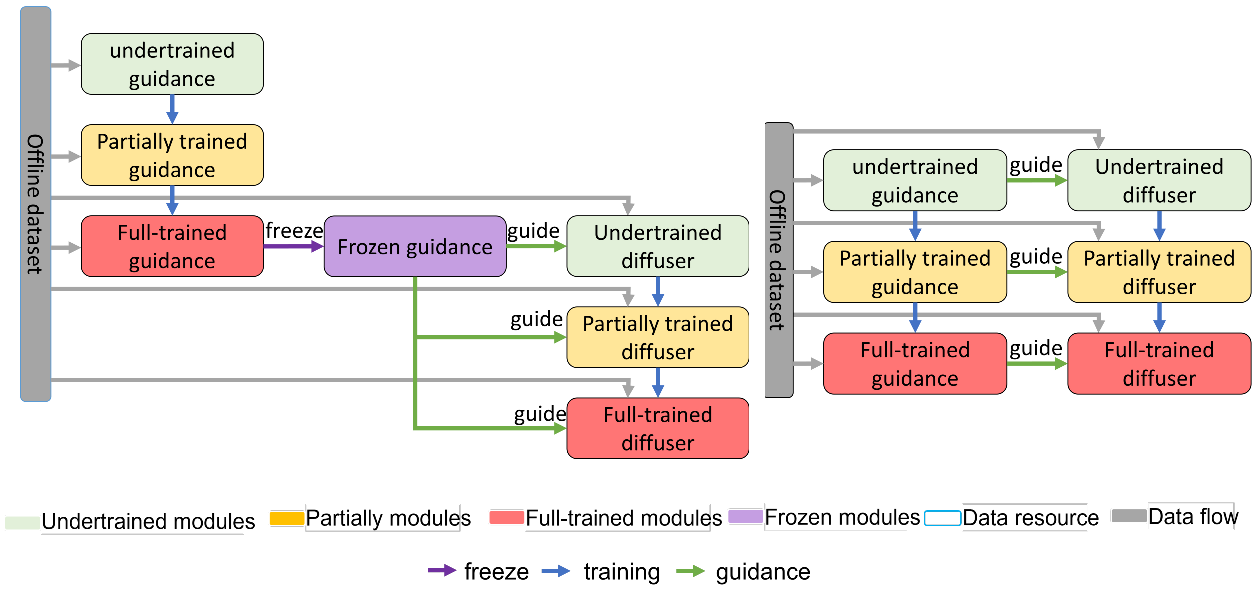


Figure 1: Comparison between the training stage(left) and inference stage(right) of GFDT

The detailed pseudo-codes of the GFDT and the traditional CFG, and a diagram comparison can be found in Appendix B.1.

2 Preliminaries

In offline reinforcement learning (RL), the objective is to learn a policy $\pi_\theta(a|s)$ that maximizes the expected return using only a fixed dataset $D = \{s, a, r, s'\}$, with interaction to the environment only at the inference stage Levine et al. (2020). When diffusion models are used as policies, several offline RL methods—including DQL Wang et al. (2023), Exponential Diffusion Process(EDP) Kang et al. (2023), and IDQL Hansen-Estruch et al. (2023)—adopt a *conditional denoising diffusion process*. The policy is defined implicitly by the reverse denoising procedure: $\hat{a}_0 \sim \pi_\theta(\cdot \mid s)$. The details of Q-Guided Diffusion are in Appendix A.

Diffusion Q-learning (DQL): The DQL model modified the traditional Q-guided diffusion and applied a variation of the Deep Q network called double Q to improve the accuracy of the Q-value estimation van

Hasselt (2010).

$$\mathcal{L}_{\pi}^{\mathrm{DQL}}(\theta) = -\mathbb{E}_{s\sim\mathcal{D}}\left[\min\left(Q_{\phi_1}(s,a^0), Q_{\phi_2}(s,a^0)\right)\right], \quad \text{where } a^0 \sim \pi_\theta(\cdot|s) \tag{1}$$

Implicit Diffusion Q-learning (IDQL). Unlike DQL, which directly incorporates the reward target into the diffusion training, IDQL adopts a two-stage design. During training, the diffusion model is updated purely via behavior cloning from the dataset, without any explicit Q-guidance. In parallel, a separate Q-value network is learned to estimate $Q(s,a)$. At inference time, the diffusion model offers candidate actions, and the guidance module evaluates them with the Q-network through a one-hot encoding scheme, and selects the action with the highest estimated Q-value. Compared to traditional Q-guided diffusion, IDQL thus applies the guidance entirely to the inference stage rather than the training stage. Although IDQL does not leverage the Q-value estimator to directly guide diffusion during training, the implementation still places the Q-network update within the same training loop as the diffusion model. This coupling is not theoretically necessary, and the consequence is a substantially higher computational and memory usage. There is a detailed analysis on the different between IDQL and GFDT in Appendix B.2. Since IDQL does not employ diffusion guidance during training, it tends to underperform on dense-reward tasks that require fine-grained, step-wise policy optimization. In contrast, it performs well on in-distribution, goal-oriented environments such as AntMaze, which are evaluated based on task success rather than on the efficiency or smoothness of the trajectory.

Efficient Diffusion Policy (EDP) introduces a key modification on traditional Q-guided diffusion: *one-step denoising*. Instead of running the full reverse chain to generate actions, EDP corrupts a dataset action a^0 to a^k in one step, and then one-step-reconstructs an approximate clean action $\hat{a}^0$ using the denoise backward path as Equation 6 in Appendix A. This *action approximation* replaces expensive sampling with a lightweight inference step, making EDP orders of magnitude faster to train. We only provide a brief summary here, and defer further details to Appendix C on the details of the one-step guidance application.

These three methods are the baselines of our methods. Currently, our modifications are typically only applied to TD-based methods and cannot be applied to Trajectory-Based methods(e.g., Diffuser Janner et al. (2022); Ajay et al. (2023)) as they use return annotations for full sequences $\tau = (s_0, a_0, ..., s_T)$. However, experiments showed low convergence rates and suboptimal precisions in their Q-value predictions, and therefore, Guidance-First Diffusion Training showed little improvement. The reason for this observation is supposed to be: when the trajectory is long, the return may be noisy due to stochastic behavior policies, so reward attribution over entire sequences can be ambiguous. (It is hard to infer which action caused the reward change.) In contrast, TD-based methods that work on (s,a) pairs avoid this issue and support more granular, local learning signals.See Appendix D for more explainations.

3 Methodology

Having introduced the GFDT method and modular module composition in the introduction, we now extend the discussion with a mathematical model to justify the approach. First, we explain why a pretrained, converged model can provide more accurate guidance to the diffusion model, better than an unconverged model that is trained jointly with it, thereby accelerating the training process. Second, we prove that as long as the guidance model can provide a direction that leads to reward improvement and the step size is small enough, it is sufficient for guiding the diffusion model, even if the guidance model was not co-trained with the diffusion model.

Theoretical Motivation. We build upon the theoretical framework established by Theorem Fujimoto et al. (2019a), which introduces Batch-Constrained Q-Learning (BCQL) as a value-based method with provable convergence guarantees under offline settings. *Given a deterministic Markov Decision Process (MDP) and coherent batch $\mathcal{B}$, along with standard Robbins-Monro convergence conditions on the learning rate, BCQL converges to $Q^{\pi_{\mathcal{B}}}(s,a)$, where $\pi^*(s) = \arg\max_{a \text{ s.t. } (s,a)\in\mathcal{B}} Q^{\pi_{\mathcal{B}}}(s,a)$. This policy is guaranteed to match or outperform any behavioral policy contained in the dataset.*

Batch-Constrained Guidance via Diffusion. Inspired by this result, we adopt a *batch-constrained guidance* framework for training diffusion policies. Our key design choice is to pretrain a guidance policy (e.g., a value function $Q(s,a)$ or behavioral prior) purely on the offline dataset, then use it to guide the sampling of a diffusion policy. Theorem 1 guarantees, the training on the offline dataset converges to an optimal reward estimation policy Q(s,a).

This optimized Q(s,a) is then added to the gradient update of the training process of the diffusion:

$$\nabla_\theta \mathcal{L}_{\text{total}} = \nabla_\theta \mathcal{L}_{\text{BC}} + \eta \nabla_\theta Q(s,a), \tag{2}$$

where $\mathcal{L}_{\text{BC}}$ is a behavior cloning loss that regularizes the learned policy to stay close to the empirical distribution of $\mathcal{B}$, and η controls the strength of the guidance. Both of these two parts are learned based on the same dataset $\mathcal{B}$, so term 1 offers regularization to avoid leaving the support of the offline dataset and term 2 encourages reward optimization.

To further ensure batch-constrained optimization behavior during action generation(inference step), we modify the reverse diffusion process by injecting Qvalue gradients of the pretrained module:

$$a_{t-1} \;\leftarrow\; f_{\text{diffusion}}(a_t, \hat{a}_0, t) \;+\; \lambda \frac{\nabla_a Q(s, \hat{a}_t)}{\|\nabla_a Q(s, \hat{a}_t)\| + \varepsilon} \;+\; \sqrt{2\tau_t}\, \xi_t,$$

where λ is a guidance coefficient. This operation encourages the final action a_0 to lie in high-reward regions. Importantly, because the value function $Q(s,a)$ is trained on the offline dataset $\mathcal{B}$, and the diffusion model itself models the same data distribution. Intuitively, in every inference step, the diffusion model generates an action, and the Q module shifts the gradient in a magnitude that is smaller than or equal to λ. So, the generated action must be inside the region of the dataset $\mathcal{B}$. Adding the small gradient shift, the entire action is still inside or close to the dataset $\mathcal{B}$'s coverage. Thus, the value-aware perturbation can be interpreted as a form of approximate, soft batch-constrained policy improvement. For a more detailed explanation, please check Appendix F

On the contrary, before sufficient training, the inaccurate gradient will misshape the diffusion distribution and prolong the training process by adding incorrect information to the optimization, which may or may not be corrected in the later training.

Admittedly, our approach does not strictly satisfy all assumptions of BCQL: the network may converge not to the true Bellman optimum but to a local minimum, an inherent limitation of non-convex optimization that cannot be fully avoided. However, we inherit its core principle: The guidance can be trained to an optimal value estimator, and any value-based guidance must be constrained to the data distribution to guarantee that the gradient guidance is reliable. This prove is also consistent with the experiment results (Section 4): in distribution guidance improves the performance of the CFG. Validated by inverse reasoning, the application scope of the GFDT method in Section 5 showed that out-of-distribution guidance cannot guarantee efficient guidance, and in practice, often jeopardizes the performance.

Theorem 2: *Joint training of the guidance module and the diffusion policy is not required. A pretrained Q-network Q_ϕ can be modularized and directly applied to guide the diffusion model, as long as Q_ϕ accurately estimates state-action values within the dataset support $\mathcal{B}$.*

We consider the perturbed diffusion sampling process after the action generation:

$$a_t = a_t + \lambda \cdot \frac{\nabla_a Q_\phi(a_t)}{\|\nabla_a Q_\phi(a_t)\|} + \sqrt{2\tau_t}\, \xi_t, \tag{3}$$

where λ is a small step size and τ_t controls the annealed noise. The gradient $\nabla_a Q_\phi$ provides a reward-sensitive vector field over the action space.

Directional Alignment and Convergence. As long as the value gradient is directionally positive correlation the diffusion model's score function(i.e., $\cos\theta = \angle(\nabla Q_\phi, \nabla \log p(a)) > 0$), then the expected change in Q_ϕ over one step is

$$\mathbb{E}[Q_\phi(a_{t+1}) - Q_\phi(a_t)] \approx \lambda \cdot \cos\theta_t \cdot \|\nabla Q_\phi(a_t)\| + \mathcal{O}(\sqrt{\tau_t}). \tag{4}$$

Moreover, since the noise is Gaussian, we have: $\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \xi_t = 0$ (in expectation), meaning the cumulative effect of noise diminishes over time, and the value gradient dominates in expectation. The action will be guided towards a rewarding-increasing direction gradually.

A pretrained Q-function can reliably steer the diffusion process toward high-reward regions without destabilizing sampling, as long as the step size is small and the guidance remains positively beneficial and smooth. This justifies the plug-and-play design of our framework and aligns with the batch-constrained principle in offline RL.

Why Pretraining Helps: Reducing Optimization Burden For a guidance model Q_ϕ trained to ϵ-accuracy on dataset $\mathcal{B}$, the gradient estimation error satisfies:

$$\mathbb{E}\|\nabla_a Q_\phi - \nabla_a Q^*\| \leq \frac{L}{\sqrt{N}} + \epsilon \tag{5}$$

Table 1: The parameters used in Equation.5

$\nabla_a Q_\phi$	Pretrained Guidance Gradient	ϵ	Final training error of Q_ϕ
L	Lipschitz constant of the Q-function class	N	Number of samples in dataset $\mathcal{B}$
$\nabla_a Q^*$	True gradient of the optimal Q-function		

Heuristically, although in reinforcement learning practice the Lipschitz constant L is rarely specified, in related areas (e.g., WGAN with spectral normalization Miyato et al. (2018)) one often enforces $L = 1$. Under this convention, Eq.5 implies that achieving a guidance accuracy of $\mathbb{E}\|\nabla_a Q_\phi - \nabla_a Q^*\| = 0.01$ would require about $N = 1/\mathbb{E}\|\nabla_a Q_\phi - \nabla_a Q^*\|^2 = 10^4$ effective samples (equivalently, $\sim 10^4$ gradient updates of the Q network). In more parameter-sensitive scenarios, pushing the accuracy further to $\mathbb{E}\|\nabla_a Q_\phi - \nabla_a Q^*\| = 0.001$ would require as many as $N = 10^6$ effective samples (i.e., 10^6 gradient updates). This illustrates the significant optimization steps that are potentially misguided before the guidance converges, and shows how our method helps reduce otherwise wasted early training of the diffusion model. Pretraining is therefore essential to ensure that guidance has semantic meaning before being applied in generation.

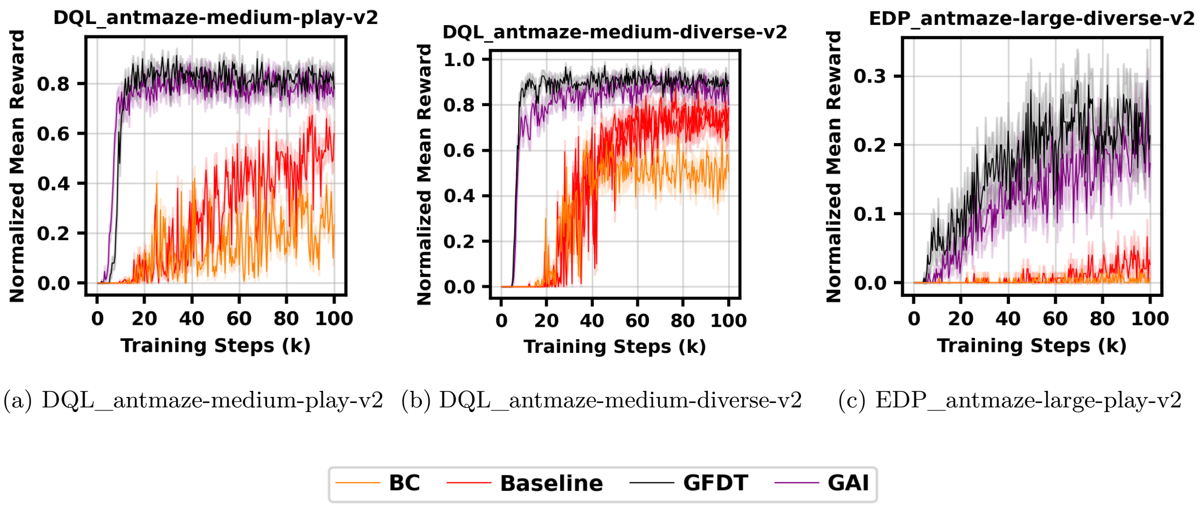
4 Experiments

We evaluate our method on standard tasks from the PyBullet D4RL benchmark Fu et al. (2021); Foundation (2024). To ensure fair comparison, we re-train three seeds of three representative diffusion-based offline RL algorithms—EDP, DQL, and IDQL(module interchange)—based on the dataset and benchmark code from Wang et al. (2024). These methods serve as illustrative examples of our theory, which can be easily extended to more diffusion methods. For more details about the environment setting, please check Appendix E. The detail comparison between our model and the baseline models is shown in Appendix G.

4.1 Performance of GFDT

All experiments were conducted using PyTorch on a high-performance computing (HPC) cluster. We adhered to any assigned framework for our work because the focus of this paper is not restricted to any specific computational framework.

From the Table 10 and Table 11, in the environments and algorithms, GFDT outperformed baseline models in 11/12 environments for DQL and 8/12 environments for EDP and . More importantly, the early performance of the GFDT is significantly better than the baseline model as shown in Table 5. On MuJoCo tasks, our method achieves an average improvement of about 2.9%, while on AntMaze tasks, the improvement ranges from 14% to 20%. This indicates that the proposed method brings consistent yet moderate gains on dense-reward locomotion tasks, and substantially larger benefits on sparse-reward, goal-directed environments.The improve of the DQL model is higher than the EDP model. Formally, DQL can be abstracted as $x_t = f(x_{t+1}, \text{guidance}_t)$ where each step depends on both the previous state x_t and a step-wise guidance signal, leading to an iterative accumulation of guidance effects. In contrast, EDP follows a single-step mapping $x_0 = f(x_T, \text{guidance})$, where the guidance is injected only once without iterative updates, which naturally limits its overall influence.



(a) DQL_antmaze-medium-play-v2 (b) DQL_antmaze-medium-diverse-v2 (c) EDP_antmaze-large-play-v2

Figure 2: **Training performance of proposed algorithms on benchmarks.**

Table 2: Comparison with standard offline RL baselines on D4RL tasks. Results ReBR, short for REBRAC from Wu et al. (2019) DICE Ma et al. (2024), CQL are taken from Kumar et al. (2020), IQL from Kostrikov et al. (2022), EDP-GFDT and DQL-GFDT, DGID and IGDD are ours.(THe abbrevations are in AppendixJ)

Env	ReBR	DICE	CQL	IQL	DQL_GFDT	EDP_GFDT	DGID	IGDD
HCME	101.1±1.5	97.3±0.6	45.3±0.3	92.7±2.8	90.2±0.4	87.2±0.0	85.61±0.01	84.77±0.00
HCMR	51.0±0.2	49.2±0.9	45.3±0.3	42.1±3.6	**67.9±0.3**	**64.4±0.0**	**70.6±0.8**	**69.8±0.0**
HCMV	65.6±1.0	60.0±0.6	46.9±0.4	50.0±0.2	**67.0±0.6**	59.3±0.4	60.9±0.9	60.9±0.2
HOME	107.0±6.4	112.2±0.3	96.6±2.6	85.5±29.7	**172.3±1.1**	**163.9±0.0**	**182.4±0.0**	**182.7±0.0**
HOMR	98.1±1.6	102.3±2.1	89.6±13.2	89.6±13.2	**153.0±0.8**	**146.4±1.7**	**157.2±0.0**	**157.2±0.0**
HOMV	102.0±1.0	100.2±3.2	61.9±6.4	65.2±4.2	**147.0±1.1**	**141.6±0.0**	**136.5±0.2**	**136.5±0.2**
WAME	111.6±0.3	114.1±0.5	104.6±1.1	112.1±0.5	**117.6±0.3**	**116.6±0.0**	**118.1±0.0**	**118.4±0.0**
WAMR	77.3±7.9	90.8±2.6	76.8±1.0	75.4±9.3	**95.6±4.8**	83.8±0.4	**93.1±0.94**	87.5±0.7
WAMV	82.5±3.6	89.3±1.3	79.5±3.2	80.7±3.4	**87.9±0.3**	84.2±0.0	87.11±0.07	87.1±0.0
L-div	54.4±25.1	91.3±3.1	14.9	27.6±7.8	90.7±5.4	31.3±6.9	33.3±6.9	46.7±7.0
L-play	60.4±26.1	85.7±4.8	15.8	42.5±6.5	**89.3±5.6**	22.7±6.7	28.0±6.7	59.3±7.0
M-div	76.3±13.5	68.6±8.6	53.7	61.7±6.1	**97.3±4.0**	52.0±7.7	58.0±7.0	68.0±6.8
M-play	84.0±4.2	72.0±6.5	61.2	64.6±4.9	**91.3±5.3**	**118.0±10.4**	56.7±7.0	74.7±6.6

Note 1 While the algorithms were proposed in the aforementioned papers, the authors did not release their raw experimental results;all numerical values for standard baselines in Table 2 are taken from Ma et al. (2023). These methods are considered as the strongest methods in offline RL.
Note 2 The left four columns show results from other methods, while the right four columns present results from our proposed method. The highest values in the left four columns are highlighted in blue. If a value in our method (right columns) exceeds the highest value on the left, it is shown in bold with a gray background.

4.1.1 The role of the guidance module

To analyze the role of reward guidance and whether the guidance is removable or replaceable, we conducted an ablation study of a series of training sessions. In this experiment, we compare the performance of GFDT, baseline models, an ablation study that removed the reward guidance part of the baseline training—behavior clone(BC), an ablation study on a behavior clone training with a guidance at the inference stage(GAI), an ablation study that pretrained the guidance module but did not freeze it in diffusion training.

Table 3: Performance comparison across tasks (DQL)

Environment	Baseline	GFDT	GAI	BC	Unfreeze
halfcheetah-expert	88.21 ± 0.28	90.43 ± 0.26	68.49 ± 1.55	85.82 ± 2.24	89.50 ± 0.23
halfcheetah-medium-expert	88.28 ± 0.53	90.18 ± 0.38	89.73 ± 0.43	85.53 ± 1.34	89.76 ± 0.61
halfcheetah-medium-replay	67.38 ± 0.34	67.93 ± 0.33	67.76 ± 0.52	55.87 ± 5.23	67.43 ± 0.37
halfcheetah-medium	59.88 ± 5.40	66.99 ± 0.60	53.36 ± 1.43	54.57 ± 4.27	55.04 ± 2.27
hopper-expert	166.76 ± 0.58	172.90 ± 0.56	162.68 ± 1.21	165.11 ± 0.67	155.15 ± 44.68
hopper-medium-expert	168.00 ± 1.18	172.31 ± 1.09	166.28 ± 1.03	166.88 ± 1.33	159.10 ± 31.75
hopper-medium-replay	151.59 ± 0.57	152.98 ± 0.75	121.97 ± 49.30	113.30 ± 35.20	150.50 ± 3.44
hopper-medium	143.92 ± 1.05	147.06 ± 1.09	71.09 ± 35.69	118.08 ± 15.41	144.59 ± 1.09
walker2d-expert	117.25 ± 0.46	120.25 ± 0.60	119.91 ± 0.52	108.05 ± 28.27	117.92 ± 0.47
walker2d-medium-expert	117.73 ± 0.35	117.63 ± 0.34	115.08 ± 1.32	117.45 ± 0.68	118.50 ± 0.73
walker2d-medium-replay	92.96 ± 0.60	95.62 ± 4.78	80.32 ± 19.52	87.14 ± 1.87	95.62 ± 4.78
walker2d-medium	87.65 ± 0.40	87.89 ± 0.27	77.33 ± 13.13	82.28 ± 0.70	87.89 ± 0.43
Average	112.47 ± 0.98	115.18 ± 0.92	99.50 ± 10.47	103.34 ± 8.10	110.92 ± 7.57
antmaze-large-diverse-v2	63.33 ± 48.19	90.67 ± 5.39	86.67 ± 5.83	66.00 ± 6.88	86.67 ± 5.83
antmaze-large-play-v2	90.00 ± 30.00	89.33 ± 5.56	88.67 ± 5.63	50.67 ± 7.07	91.33 ± 5.30
antmaze-medium-diverse-v2	93.33 ± 24.94	97.33 ± 4.01	94.00 ± 4.87	78.00 ± 6.44	96.00 ± 4.43
antmaze-medium-play-v2	67.33 ± 46.90	91.33 ± 5.30	88.67 ± 5.63	68.00 ± 6.83	93.33 ± 4.99
Average	78.50 ± 37.51	92.17 ± 5.07	89.50 ± 5.49	65.67 ± 6.80	91.83 ± 5.14

Table 4: Performance comparison across tasks (EDP)

Environment	EDP_baseline	GFDT	GAI	BC	Unfreeze
halfcheetah-expert	86.55 ± 0.00	86.82 ± 0.00	78.78 ± 0.52	85.82 ± 2.24	85.98 ± 0.00
halfcheetah-medium-expert	86.74 ± 0.00	87.16 ± 0.00	78.18 ± 0.15	85.53 ± 1.34	86.85 ± 0.01
halfcheetah-medium-replay	65.78 ± 0.03	64.42 ± 0.03	47.08 ± 1.90	55.87 ± 5.23	64.42 ± 0.03
halfcheetah-medium	54.59 ± 0.04	59.26 ± 0.39	53.56 ± 0.02	54.57 ± 4.27	55.33 ± 0.11
hopper-expert	161.23 ± 0.02	163.65 ± 0.02	151.93 ± 3.23	165.11 ± 0.67	162.24 ± 0.01
hopper-medium-expert	161.36 ± 0.02	163.95 ± 0.01	141.07 ± 10.51	166.88 ± 1.33	163.68 ± 0.01
hopper-medium-replay	112.16 ± 2.83	139.75 ± 6.69	114.32 ± 3.26	113.30 ± 35.20	119.91 ± 11.62
hopper-medium	141.16 ± 0.02	141.64 ± 0.01	83.25 ± 6.63	118.08 ± 15.41	141.54 ± 0.05
walker2d-expert	116.14 ± 0.00	116.34 ± 0.00	114.27 ± 0.01	108.05 ± 28.27	116.36 ± 0.00
walker2d-medium-expert	116.31 ± 0.01	116.59 ± 0.00	109.55 ± 3.05	117.45 ± 0.68	117.07 ± 0.00
walker2d-medium-replay	85.02 ± 0.01	83.77 ± 0.37	68.84 ± 3.45	87.14 ± 1.87	85.19 ± 0.02
walker2d-medium	84.24 ± 0.00	84.21 ± 0.00	58.21 ± 5.29	82.28 ± 0.70	84.44 ± 0.00
Average	105.94 ± 0.25	108.96 ± 0.63	91.59 ± 3.17	103.34 ± 8.10	106.92 ± 0.99
antmaze-large-diverse-v2	30.67 ± 6.99	31.33 ± 6.91	28.67 ± 46.67	10.67 ± 5.56	47.33 ± 7.25
antmaze-large-play-v2	21.33 ± 6.40	22.67 ± 6.70	18.67 ± 40.64	18.67 ± 6.24	25.33 ± 6.82
antmaze-medium-diverse-v2	67.33 ± 9.06	52.00 ± 7.72	2.00 ± 14.00	18.00 ± 6.46	13.33 ± 6.14
antmaze-medium-play-v2	73.30 ± 10.34	118.00 ± 10.35	90.00 ± 95.74	76.00 ± 9.28	113.33 ± 9.89
Average	48.16 ± 8.20	56.00 ± 7.92	34.83 ± 49.26	30.83 ± 6.89	49.83 ± 7.52

-Beyond the superior performance of our proposed algorithm, we further investigate its other properties through the ablation Study. The pretrained but non-frozen guidance model in diffusion training outperforms the baseline but does not surpass GFDT, confirming that additional training of the guidance model is unnecessary. The slight performance drop is likely caused by overfitting of the guidance model under over-training. GAI, removing the guidance module, results (in training) got worse performance and higher variance, underscoring the important role of guidance. Finally, we also tested that random guidance leads to complete failure of the diffusion model, highlighting that inaccurate guidance signals can be detrimental. This supports our claim that a valid guidance model is critical for successful training.

Table 5: Training gradient steps to get 95% performance and parameter statistics summary (batch size 256)

Env	DQL	GFDT_DQL	EDP	GFDT_EDP	DDIG	IDDG
HCEX	7600	2800 (36.84%)	18000	3600 (20.00%)	13200 (63.46%)	10400 (50.00%)
HCME	16400	2800 (17.07%)	31200	6000 (19.23%)	26400 (81.48%)	14400 (44.44%)
HCMR	8000	3200 (40.00%)	20400	8400 (41.18%)	7200 (26.47%)	35600 (130.90%)
HCM	52000	7600 (14.62%)	10800	10000 (92.59%)	6400 (7.24%)	45600 (51.60%)
HOEX	30800	15200 (49.35%)	8400	8400 (100.00%)	38400 (118.52%)	4400 (13.60%)
HOME	41600	23600 (56.73%)	24000	16800 (70.00%)	44400 (105.71%)	56000 (133.30%)
HOMR	24800	3200 (12.90%)	40800	52800 (129.41%)	71200 (127.14%)	35600 (63.60%)
HOM	79200	12800 (16.16%)	50400	76800 (152.38%)	90800 (76.95%)	19200 (16.30%)
WAEX	59600	14000 (23.49%)	10800	7200 (66.67%)	51200 (69.57%)	30400 (41.30%)
WAME	96000	72400 (75.42%)	21600	18000 (83.33%)	47600 (42.65%)	31600 (28.30%)
WAMR	26000	9600 (36.92%)	38400	10800 (28.13%)	10800 (16.07%)	400 (0.60%)
WAM	10800	3600 (33.33%)	79200	51600 (65.15%)	24000 (55.05%)	60400 (138.50%)
AVG	37733	14233 (34.40%)	29500	22533 (72.34%)	35967 (65.86%)	28667 (59.37%)
Ldiv	500000	47600 (9.52%)	1600000	900000 (56.25%)	64800 (12.96%)	400000 (80.00%)
Lplay	1300000	29200 (2.25%)	1300000	87600 (6.74%)	76400 (5.88%)	70800 (5.45%)
Mdiv	1500000	18000 (1.20%)	800000	17200 (2.15%)	37600 (2.51%)	44400 (2.96%)
Mplay	89600	15200 (16.96%)	1400000	1400000 (100.00%)	85200 (95.09%)	43200 (48.21%)
AVG	847400	27500 (7.48%)	1275000	601200 (41.28%)	66000 (29.11%)	139600 (34.16%)

Table 6: Parameter statistics summary for different model widths. (This model has a hidden layer size of 256.) Explanation of this table are in Appendix**??**.

	128	**256**	**512**
Diffusion Params	172,230 (70.0%)	174,534 (38.1%)	172,230 (13.7%)
Guidance Params	73,986 (30.0%)	283,650 (61.9%)	1,082,370 (86.3%)
Total Memory	0.94 MB	1.75 MB	4.79 MB

The training time of the diffusion model decreased significantly as shown in Table.5 with decreased computational resources—while they have the same training for the guidance module, the training steps for the diffusion module was significantly decreased. The following sections discuss how to plug and play a pretrained module into the diffusion model, making the modules reusable.

In Table 6, we analyze the number of parameters in the guidance module and the diffusion model. The parameter counts objectively reflect the memory footprint and computational cost of each component. Since we pre-train the guidance module, the peak memory usage during GFDT training only needs to accommodate the largest of these components. Therefore, we consider the effective reduction to be determined by the smaller of the two overlapping components, which corresponds to a 38.1% decrease for our model.

4.2 Plug-and-Play Model Composition

Our research reveals that diffusion models exhibit unique plug-and-play compatibility with their guidance modules, following the theoretical proof. We evaluate two hybrid configurations without any additional training: 1) DQL-as-Guidance + IDQL-as-Diffusion(IDDG), 2) IDQL-as-Guidance + DQL-as-Diffusion(DDIG). as shown in Figure 3, both combinations: (1) achieve final performance comparable to the DQL baseline, the higher one of these two models in Mujoco environments. (2)They performed lower results in Antmaze environments because the Plug and Play method does not match with the Max Q algorithm that is applied in Antmaze environments (3) They exhibited significant faster initial convergence speeds compared to baselines, and (4) maintain stability despite architectural misplacement.The detailed analysis of this ablation study is in the Appendix.H. The key point it that for environments with dense reward(Mujoco), the discrepancy or mismatch can be compensated and corrected immediately, the module interachange is applicable and even beneficial for optimality and for smoothing the adjustments. However, for sparse rewarding environments(Antmaze) that need a high precision, the error can accumulate and the reward estimation becomes out-of-distribution, because of the mismatch.

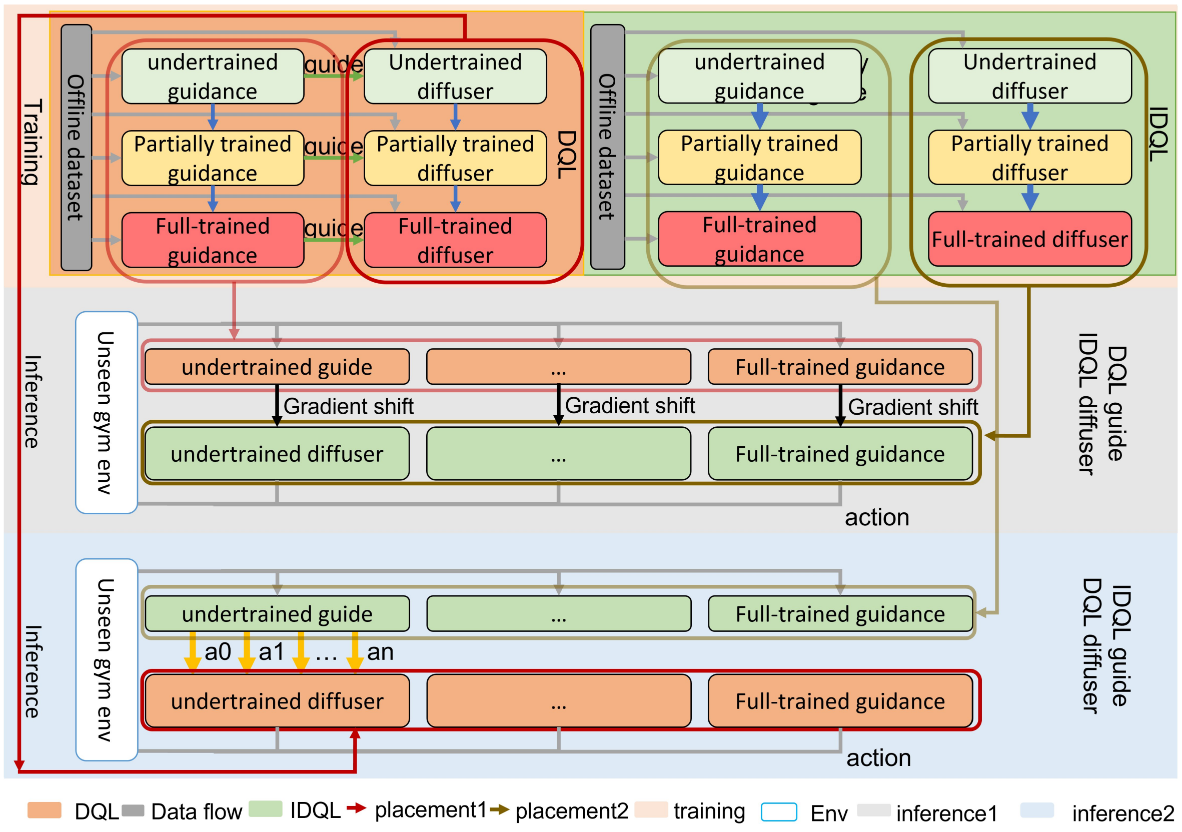


Figure 3: Diagram of plug-and-play modular configuration

Most importantly, despite the absence of joint training or any task-specific fine-tuning, the plug-and-play configurations achieve performance comparable to established baselines such as CQL and IQL. Although their results are slightly below the best-performing methods (e.g., DICE), the fact that two independently trained modules can be directly combined to reach this level of performance highlights the strong modular compatibility and generalization potential of our framework. This result implies that: Guidance modules can provide effective policy improvement signals regardless of diffusion model architecture with proper setup. It implies that the normalizing step in the training and inference parts is expected to be an important reason why these two modules are interchangeable, as explained in the Methodology Section Theorem 2. Finally, early-stage advantages suggest that when the modules are both under-trained and are not perfect, a differently structured model is not only unharmful, but can cross-correct with the errors of each module. We find that a guidance module—trained independently and even with a completely different architecture—can be directly applied to a diffusion model. This experiment is consistent with our theoretical analysis that, as long as the guidance model provides a reliable estimate of the reward(gradient estimate per se), it can be reused across structurally dissimilar models. Such modularity enables flexible training pipelines and paves the way for reusable, task-agnostic reinforcement learning components. Finally, as we have mentioned in the theoretical part, if the guidance model is not independent trained on the same data distribution, there is no guarantee that the independent module will be beneficial. In the following section, we illustrate that pretraining on out-of-distribution(OOD) data is more likely to be detrimental than beneficial in practice.

5 Application scope and Limitations

To illustrate the distribution shift between datasets, We performed PCA analysis on the datasets and plotted the region that contains 60% of the data points to eliminate the instability caused by outliers. The highlighted points are the top 100 highest-reward points. The positions of these high-reward points varied significantly

Table 7: Performance comparison with standard deviation in ± format (2 decimal places)

Dataset	**DQL**	**IDQL**	DGID	IGDD
halfcheetah-expert	88.21±0.28	71.07±0.02	85.59±6.92	81.64±0.86
halfcheetah-medium-expert	88.28±0.53	72.03±0.01	85.61±0.01	84.77±0.00
halfcheetah-medium-replay	67.38±0.34	55.90±0.00	70.58±0.84	69.79±0.01
halfcheetah-medium	59.88±5.40	52.70±0.00	60.87±0.89	60.87±0.23
hopper-expert	166.76±0.58	160.53±0.00	159.18±4.11	208.67±0.02
hopper-medium-expert	168.00±1.18	128.25±2.82	182.47±0.03	182.68±0.02
hopper-medium-replay	151.59±0.57	55.53±0.84	157.24±0.00	157.24±0.00
hopper-medium	143.92±1.05	65.94±0.26	136.53±0.23	136.53±0.20
walker2d-expert	117.25±0.46	113.78±0.00	116.64±0.14	116.64±0.02
walker2d-medium-expert	117.73±0.35	70.79±0.02	118.11±0.00	118.39±0.00
walker2d-medium-replay	92.96±0.60	71.83±0.05	93.10±0.94	87.35±0.07
walker2d-medium	87.65±0.40	66.78±0.01	87.11±0.07	87.11±0.01
Subset Avg	112.47±0.98	82.09±0.34	112.75±1.18	115.97±0.12
large-diverse	51.33±7.07	53.33±7.16	33.33±6.87	46.67±7.06
large-play	90.00±5.48	62.00±7.79	28.00±6.70	59.33±7.01
medium-diverse	93.33±4.99	82.67±7.98	58.00±7.03	68.00±6.83
medium-play	67.33±6.85	56.00±6.58	56.67±7.04	74.67±6.59
Full Avg	75.50±6.10	63.50±7.38	44.00±6.91	62.17±6.87

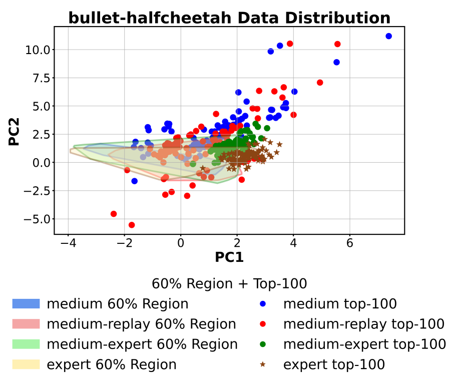
Table 8: Default DQl with Different Guidance Strengths Experimental Results for Expert Adroit

	relocate		pen		hammer		door	
	mean	std	mean	std	mean	std	mean	std
η=0(BC)	105.9296	±0.843	144.8144	±5.58	129.2588	±1.261	105.9296	±0.843
η=0.05	105.4705	±0.906	141.9434	±6.13	128.5337	±1.273	105.4705	±0.906
η=0.1	105.6861	±1.049	130.3630	±7.05	129.2724	±1.091	105.6861	±1.049
η=1	-0.2692	±0.2	36.6698	±7.8345	0.1452	±2.2361	-0.1484	±1.4142

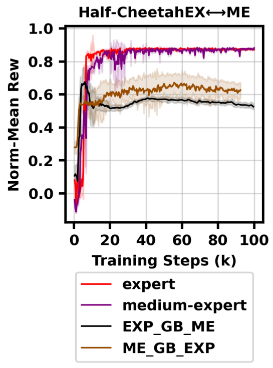
across datasets, even within the same environment and the shifting trade suggests the gradual converging process. The differences in the plot imply that for any given dataset, the data from another expertise level often lies in an OOD region, as shown in Table.4. The dataset distributions of other environments are show in Appendix I. Based on this observation, we conducted OOD experiments by training a guidance network on one dataset, freezing it, and then using it to guide the training of a main diffusion model on a different dataset. Across 36 tested models, 28/36(78%) exhibited significant performance degradation. Among the eight cases with good performance, half benefited from guidance modules pretrained using medium replay models, which cover a broader range of data and thus generalize better than other datasets.

From these results, we conclude that the decouple method is highly sensitive to distribution shift. Training a guidance network on in-distribution data is critical for success. Otherwise, even expert-level data cannot reliably guide training on a different distribution. This ablation study is consistent with the conclusion in the Methodology part that OOD pretraining cannot guarantee performance improvement.

Another kind of limitation of Q-guided diffusion, not only in our methods but also, is whether the guidance should exist in the training. As shown in Table 8, for high-dimensional tasks with narrow data coverage (e.g., Adroit), even slight guidance can push the policy out-of-distribution, causing performance to drop significantly. In these cases, relying solely on behavioral cloning proves more stable and effective. Conversely, for tasks with moderate difficulty and abundant data, Q-guidance can successfully enhance performance.



(a) Data distribution



(b) Cross experiment expert & median Expert

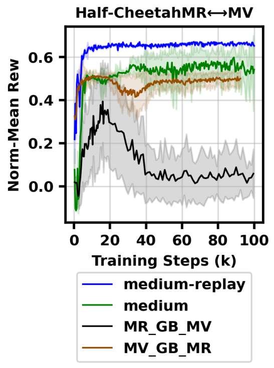


Figure 4: Cross-training of different data levels

6 Related Work

Modular and Decoupled Training: Modular training has long been pursued as a desirable paradigm. In the context of diffusion models, the earliest attempts at modularity can be traced to classifier guidance Dhariwal & Nichol (2021), where a pretrained diffusion model was paired with a separately trained classifier to steer sampling. This approach was unstably designed to amplify the possibility of one class while suppressing all others, ignoring that features are often shared across categories, leading to distorted and fragile guidance. Recently, energy-based guidance Lu et al. (2023) was proposed, where a diffusion model is trained first and an energy model is subsequently learned to provide guidance. However, such post-hoc modularization has proven fragile in practice—e.g., in our own experiments, more than half of the runs diverged. In contrast, our method inverts this order: we first train a guidance module using supervised learning from offline data, and then use this frozen module to learn a diffusion. It serves as a general-purpose enhancement to existing architectures of CFG. Another line of research is semi-modularized: although IQL can be seen as a modular paradigm—learning Q-values first and then selecting one-hot action in the inference stage—it does not apply guidance during training, and its performance often lags behind joint methods such as DQL. The importance of stability under distributional shift has been repeatably mention in classic offline RL algorithms such as BCQ Fujimoto et al. (2019b), CQL Kumar et al. (2020), and BRAC Wu et al. (2019), our work proposes a guidance-first modular framework that enhances training in offline RL can also be considered as a batch-constrained or conservative regularization, using the pretrained guidance to regularize diffusion process. Finally, modular training is valid, as on vision or text diffusion-based generation widely exists; large language models and vision language models have Retrieval Augmented Generations or other special models separately trained as building blocks; however, since these modules use the web-harvested data, the limitations of OOD are not well-discussed. Offline RL must contend with severe out-of-distribution issues, making modularization brittle. This challenge has been extensively studied, with methods such as BRAC Wu et al. (2019), CQL Kumar et al. (2020), MOPO Yu et al. (2020), and AWAC Nair et al. (2020) proposing different strategies to mitigate extrapolation error. Our study provides the first systematic examination of when modular guidance is feasible and demonstrates that, under the challenges of offline RL, modular training can succeed if designed around guidance-first principles.

Plug-and-Play Modular Composition. The idea of plug-and-play composability is to treat pretrained modules—originally developed for other purposes—as reusable building blocks. Such composability is rarely studied and highly empirical. We propose that plug-and-play composition requires distributional alignment. For instance, in diffusion models, CLIP-based guidance Nichol et al. (2021); Ramesh et al. (2021) applied a pretrained vision–language model as guidance, but fails on noisy intermediate states, where distribution mismatch undermines compositionality. Circumventions exist: some prior works explore modular policy composition Andreas et al. (2017); Peng et al. (2019), focusing on skill chaining or subpolicy selection. These methods apply the plug-ins as subproblem solvers in heuristic models instead of direct reward guidance. A common challenge concerns **I/O calibration** between modules' signal magnitude.

This issue is acute in systems with complex plug-in connections, such as adapter-based methods Mou et al. (2023); Zhang et al. (2023); Ye et al. (2023), which rely heavily on the backbone's feature space. When transferred across architectures (e.g., from SD1.5 to SDXL), their performance collapses due to representational mismatch, showing these adapters are not universal interfaces. The same logic applies in NLP, where Retrieval-Augmented Generation (RAG) Lewis et al. (2020) often suffers when the retriever is misaligned. Related retrieval-augmented frameworks such as REALM Guu et al. (2020), FiD Izacard & Grave (2021), and Atlas Izacard et al. (2022) further highlight the need for careful design of retriever–generator alignment. Overall, the design of plug-and-play models requires careful consideration to ensure alignment and stability.

Finally, plug-and-play can be beneficial if properly implemented. Value-based reinforcement learning methods often suffer from intrinsic bias, since a single network trained on limited data can easily over- or under-estimate values in unseen regions. A classical remedy is *Double Q-learning* **??**, which decouples action selection and evaluation using two networks. Our work follows this family of ideas: inspired by Double Q-learning and ensemble learning, we leverage two independently initialized modules to cancel stochastic biases inherent in a single model, a technique widely adopted in reinforcement learning but underexplored in diffusion-based policies.

7 Future work

In practice, we observe that during the early stage of training, the reconstruction loss overwhelmingly dominates the overall objective, effectively outnumbering the guidance loss. As a result, for roughly the first twenty thousand gradient steps, the model primarily focuses on reconstruction, and the influence of guidance remains negligible. Once the reconstruction error has sufficiently decreased, the effect of guidance becomes more visible—its gradients start shaping the diffusion behavior toward higher-reward regions.

However, we also notice that in some cases both in the baselines and in GFDT, applying a guidance signal can lead to unstable training behavior, occasionally causing a failed training performance collapse after initial improvement. The inverse V shape curve in Fig.4 is a good example when the guidance is not applied properly. However, OOD guidance issue is not intrinsic, but occasional failing can be fixed. A simple and effective remedy is to gradually introduce the guidance term—for example, linearly increasing its weight after the first few thousand training steps and finally applying the full weight to the model—which empirically stabilizes training and consistently prevents such failures. In our experiments, after applying this trick, training failure has never occurred.

Reproducibility and Ethics Statement

Experiments use three random seeds; hyperparameters are in Appendix E. Code, pretrained models, and scripts are available at `https://github.com/modulardiffusion-design/Modular_diffusion`. The study uses public benchmarks only and has no foreseeable negative societal impact, though safe and fair deployment should be considered.

A Q-Guided Diffusion

Q-Guided DiffusionJanner et al. (2022) has a forward noising process and a reverse denoising process. The forward noising process gradually perturbs a clean action a_0 into a noisy version a_k:

$$a_k = \sqrt{\bar{\alpha}_k}\, a_0 + \sqrt{1-\bar{\alpha}_k}\,\epsilon, \quad \epsilon \sim \mathcal{N}(0, I), \quad \bar{\alpha}_k = \prod_{i=1}^{k} \alpha_i,$$

where $k \in \{1, \dots, T\}$ denotes the diffusion timestep (or noise level), and $\bar{\alpha}_k$ is the cumulative product of noise scheduling coefficients . A denoising network $\epsilon_\theta(a_k, k, s)$ is trained to recover the clean action a_0 from its noisy counterpart $a_k \sim \mathcal{N}(0, I)$. During reverse diffusion, we can form an estimate of the clean action:

$$\hat{a}_0 = \frac{1}{\sqrt{\bar{\alpha}_k}}\Big(a_k - \sqrt{1-\bar{\alpha}_k} \cdot \epsilon_\theta(a_k, k, s)\Big). \tag{6}$$

The behavior clone loss is formulated as: $\mathcal{L}_{\text{BC}} = \mathbb{E}_{a_0,\epsilon,t}[\|\epsilon_\theta(a_t, t, s) - \epsilon\|^2]$. To incorporate reward information, Q-guided diffusion introduces an actor loss $\mathcal{L}_Q$, and the entire loss becomes: $\mathcal{L}_{\text{actor}} = \mathcal{L}_{\text{BC}} + \eta\,\mathcal{L}_Q$, which is the sum of the behavior cloning loss, and Q-loss, which is controlled by the strength of η. Here $Q_\phi(s, a)$ denotes the learned Q-function, i.e., an estimation of the expected discounted return starting from state s and action a: $Q_\varphi(s,a) \approx \mathbb{E}[\sum_{t=0}^{\infty} \gamma r_t \,\Big|\, s_0 = s, a_0 = a]$, $\quad$ (γ is the discount factor, r_t the reward at step t). For stable guidance, we normalize this value by its expectation over sampled actions, defining $\tilde{Q}_\phi(s,a) = Q_\phi(s,a)/\mathbb{E}_a[Q_\phi(s,a)]$. This normalization ensures scale invariance of the guidance signal and enables interchangeable use of guidance modules across actor-critic variants (e.g., DQL and IDQL) without a mismatch in magnitude.(A more detailed derivation is provided in section3.)

A similar design also exists at inference time, the Q-network provides explicit guidance by adjusting the sampled actions:

$$\tilde{a}_0 = a_0 + \lambda \frac{\nabla_a Q_\phi(s, a_0)}{\|\nabla_a Q_\phi(s, a_0)\| + \epsilon}, \tag{7}$$

where λ balances exploitation of the learned Q-function against fidelity to the diffusion. The normalizing process is inherent from Ho et al. (2020) and is expected to be important in the modular and plug-and-play designs.

B Algorithmic Details

B.1 GFDT Algorithmic Details

The entire training and inference process of GFDT is shown in Fig.5, as well as the algorithm1.

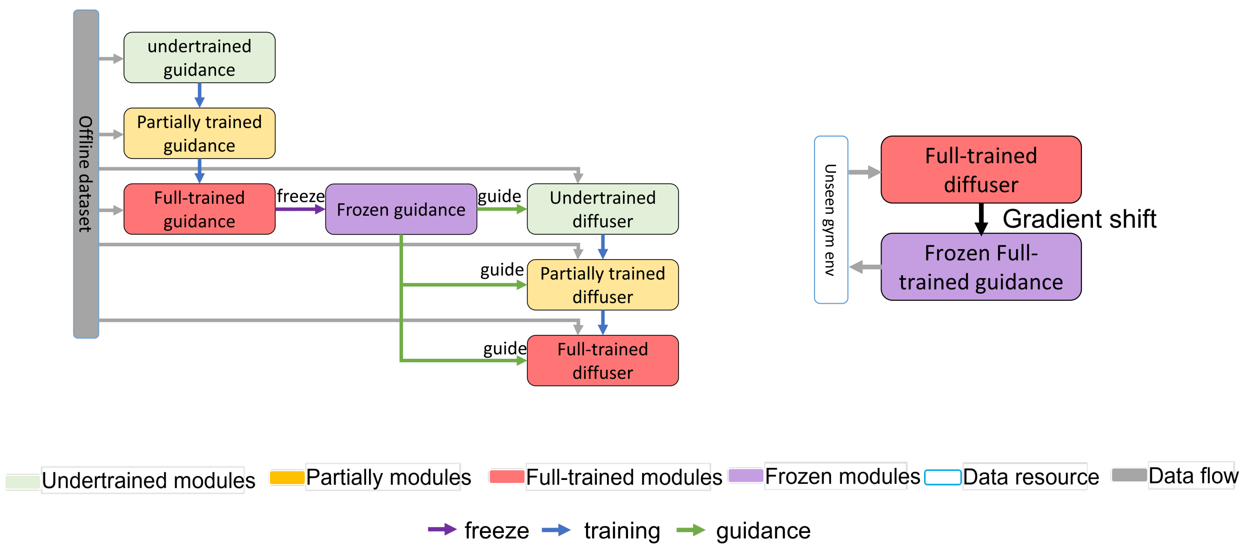


Figure 5: Training and inference stage of GFDT

Algorithm 1 Guidance-First Diffusion Training (GDFT)

```
Require: Offline dataset $\mathcal{D} = \{(s, a, r, s')\}$, Q-network $Q_\phi$, diffusion model $\epsilon_\theta$, guidance scale $\lambda$, training steps $N_q$, $N_\theta$
 1: // Step 1: Train Q-function (guidance model)
 2: for $i = 1$ to $N_q$ do
 3:     Sample minibatch $(s, a, r, s') \sim \mathcal{D}$
 4:     TD target: $y \leftarrow r + \gamma \cdot \max_{a'} Q_\phi(s', a')$
 5:     Update $Q_\phi$ to minimize $\mathcal{L}_Q = \|Q_\phi(s, a) - y\|^2$
 6: Freeze $Q_\phi$
 7: // Step 2: Train diffusion model with frozen guidance
 8: for $j = 1$ to $N_\theta$ do
 9:     Sample $(s, a_0) \sim \mathcal{D}$
10:     Add noise: $a_t \leftarrow \sqrt{\bar{\alpha}_t} a_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon$, $\epsilon \sim \mathcal{N}(0, I)$
11:     Predict noise: $\hat{\epsilon} \leftarrow \epsilon_\theta(a_t, s, t)$
12:     Update $\epsilon_\theta$ to minimize $\mathcal{L}_{\text{diff}} = \|\hat{\epsilon} - \epsilon\|^2 + \mathcal{L}_Q$
                                                        ▷ $Q_\phi$ frozen; $\mathcal{L}_Q$ still updates $\epsilon_\theta$
13: Return trained $\epsilon_\theta$, frozen $Q_\phi$
14:
15: Inference:
16: for denoise steps do
17:     Sample candidate $a_k \sim \pi_\theta(\cdot|s)$
18:     Apply guidance: $a_{k-1} \leftarrow a_k + \lambda \cdot \nabla_a Q_\phi(s, a_k)$
```

Important explanation of the frozen Q network still updating the diffusion network The details of GFDT have been thoroughly explained in the main content. A reasonable concern is the role of Q_ϕ. Although Q_ϕ is frozen and does not update the parameters of the guidance network, its output values are still used to adjust the parameters of the diffusion network, encouraging it to generate samples with higher Q values.

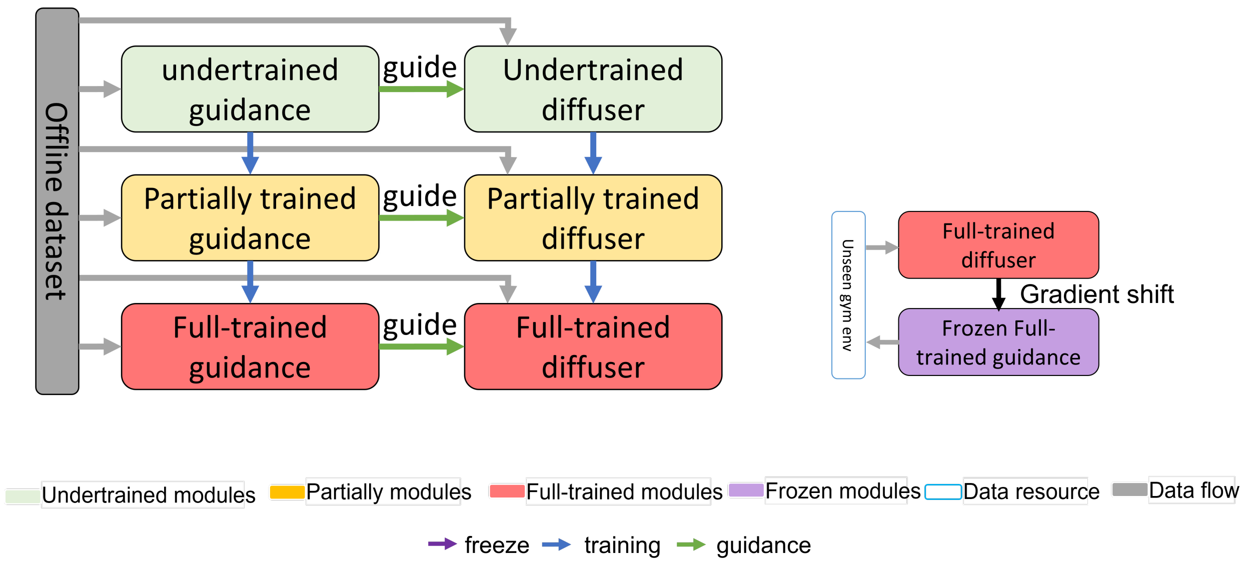


Figure 6: Training and inference stage of traditional CFG

Algorithm 2 Traditional Gradient Guidance Training

```
Require: Offline dataset $\mathcal{D} = \{(s, a, r, s')\}$, diffusion model $\epsilon_\theta$, Q-function $Q_\phi$ (learned jointly), training
    steps $N_\theta$
 1: for $j = 1$ to $N_\theta$ do
 2:     Sample $(s, a_0, r, s') \sim \mathcal{D}$
 3:     TD target: $y \leftarrow r + \gamma \cdot \max_{a'} Q_\phi(s', a')$
 4:     Update $Q_\phi$ to minimize $\mathcal{L}_Q = \|Q_\phi(s, a_0) - y\|^2$
 5:     Add noise: $a_t \leftarrow \sqrt{\bar{\alpha}_t}\, a_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$
 6:     Predict noise: $\hat{\epsilon} \leftarrow \epsilon_\theta(a_t, s, t)$
 7:     Update $\epsilon_\theta$ to minimize $\mathcal{L}_{\text{diff}} = \|\hat{\epsilon} - \epsilon\|^2 + \mathcal{L}_Q$
 8: Return trained $\epsilon_\theta$, jointly-trained $Q_\phi$
 9:
10: Inference:
11: for denoise steps do
12:     Sample candidate $a_k \sim \pi_\theta(\cdot|s)$
13:     Apply guidance: $a_{k-1} \leftarrow a_k + \lambda \cdot \nabla_a Q_\phi(s, a_k)$
14: Return $a^\star$
```

B.2 IDQL Algorithmic Details

Training. Implicit Diffusion Q-learning (IDQL) decouples the training of the diffusion model from the Q-value estimator. The diffusion policy $\pi_\theta(a|s)$ is trained purely by behavior cloning (BC) from the offline dataset $\mathcal{D} = \{(s, a)\}$, i.e.,

$$\min_\theta \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[\|a - \pi_\theta(s)\|^2\right], \tag{8}$$

without any reward or Q-guidance incorporated into the diffusion process. In parallel, a separate Q-network $Q_\phi(s, a)$ is learned by standard temporal-difference (TD) regression:

$$\min_\phi \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}} \left[\left(Q_\phi(s, a) - (r + \gamma \max_{a'} Q_\phi(s', a'))\right)^2\right]. \tag{9}$$

Notably, the Q-network is updated together with the diffusion model during training, although it does not directly affect the diffusion optimization.

Inference. At deployment, the diffusion model generates n candidate actions $\{a_1, a_2, \ldots, a_n\}$ sampled from the diffusion model $\pi_\theta(\cdot|s)$. These are then evaluated with the learned Q-network, and the action with the highest Q-value is selected:

$$a^\star = \arg \max_{i=1,\ldots,K} Q_\phi(s, a_i). \tag{10}$$

This procedure can also be interpreted as a one-hot selection mechanism over the candidate actions, where the Q-network acts as a ranking function. The algorithm of IDQL is in

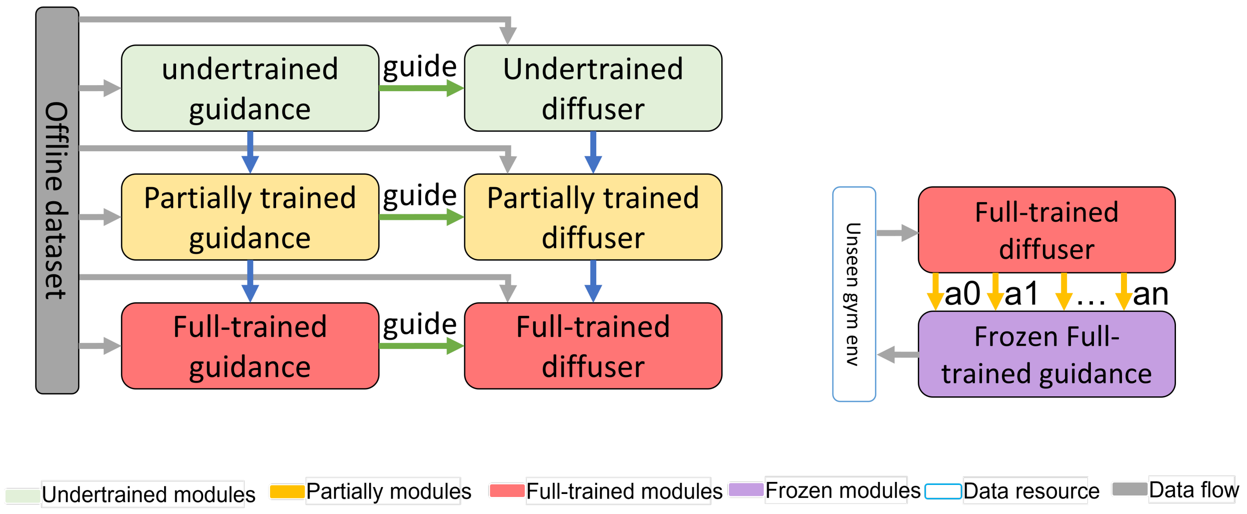


Figure 7: Training and inference stage of traditional CFG.

Algorithm 3 Implicit Diffusion Q-learning (IDQL)

```
1:  Input: offline dataset $\mathcal{D}$
2:  Initialize diffusion policy $\pi_\theta$ and Q-network $Q_\phi$
3:  while not converged do
4:      Sample batch $(s, a, r, s')$ from $\mathcal{D}$
5:      Update $\pi_\theta$ by behavior cloning on $(s, a)$
6:      Update $Q_\phi$ by TD regression on $(s, a, r, s')$

7:  Inference:
8:  Input: state $s$
9:  for $k = K, K-1, \ldots, 1$ do
10:     for $n = 1, \ldots, N$ do
11:         Sample candidate action $a_n^{(k)} \sim \pi_\theta(\cdot \mid s)$
12:         Evaluate $Q_\phi(s, a_n^{(k)})$
13:     Select best action at step $k$:
            $a^{(k)\star} = \arg \max_{n=1,\ldots,N} Q_\phi(s, a_n^{(k)})$
14: Return $a^{(1)\star}$
```

Comparison between GFDT and IDQL The key distinction between IDQL and our proposed GFDT lies in how Q-guidance is incorporated. In IDQL, a separate Q-guidance network is trained, but reward information is not injected into the diffusion model during training. This design makes IDQL relatively stable under the distribution, since the learned policy is not directly biased by Q-values that could otherwise push the distribution toward out-of-distribution regions. However, the downside is that the policy is less reward-optimal, because it is guided primarily by behavioral cloning rather than directly exploiting reward signals. At inference time, IDQL applies reward guidance only as a one-hot selection among the generated actions. While this procedure ensures the selected actions are rewarding, they remain restricted to the support of the behaviorally cloned generator, which limits the achievable performance compared to gradient-guided methods.

In contrast, GFDT explicitly integrates Q-guidance into the generative diffusion process. Reward information actively shapes the learned distribution during training, which leads to more reward-aligned behaviors. At inference time, GFDT further applies a gradient shift, using gradient descent to nudge the generated action toward the locally optimal point. As a result, GFDT consistently outperforms both the baseline CFG and IDQL in experiments. Although gradient-based guidance has the potential to destabilize training, normalization and a staged training scheme mitigate this risk: reconstruction dominates in the early phase and Q-guidance gradually takes effect afterwards, and therefore, destabilized training can manifest a certain V-shape mode. (see Appendix I).

C Efficient Diffusion Policy (EDP)

Efficient Diffusion Policy (EDP) is a variant of Q-guided diffusion that primarily improves computational efficiency. Its central idea is **one-step denoising**, which bypasses the expensive multi-step reverse process used in standard diffusion policies.

Standard Diffusion vs. One-step Denoising In traditional diffusion policies, generating a clean action a^0 requires running a long reverse chain:

$$a^T \to a^{T-1} \to \cdots \to a^0,$$

where T is typically large (e.g., 100–1000). This iterative procedure is computationally expensive.

Instead, EDP corrupts a dataset action a^0 directly into a noisy action a^k in one step:

$$a^k = \sqrt{\bar{\alpha}_k}\, a^0 + \sqrt{1-\bar{\alpha}_k}\, \epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \tag{11}$$

Then, it reconstructs an approximate clean action $\hat{a}^0$ by applying the denoiser once:

$$\hat{a}^0 = \frac{1}{\sqrt{\bar{\alpha}_k}}\, a^k - \frac{\sqrt{1-\bar{\alpha}_k}}{\sqrt{\bar{\alpha}_k}} \cdot \epsilon_\theta(a^k, k, s), \tag{12}$$

where ϵ_θ is the learned denoising network conditioned on state s.

Role of Q-guidance Since the one-step approximation introduces bias, EDP relies more strongly on the Q-function to refine $\hat{a}^0$. In practice, the algorithm evaluates candidate actions using Q-values and adjusts them toward high-value regions, which compensates for the reduced accuracy of the denoising step.

The main differences are 1)Cost reduction: iterative sampling ($O(T)$ steps) $\to$ one-step approximation ($O(1)$). 2)Trade-off:less precise denoising $\to$ stronger dependence on Q-guidance. 3)Outcome: training time reduced by orders of magnitude (days $\to$ hours), at the expense of slightly noisier reconstructions. In summary, EDP is an acceleration of traditional CFG diffusion.

D Trajectory-based Guidance: Why it Fails in Offline RL

This algorithm currently does not apply to trajectory-based methods. We trained several trajectory-return guided variants; however, the guidance estimates did not *converge* in the actionable sense: they moved from near-zero to a coarse range (e.g., $200-300$ in a median expert dataset with Q values of $\approx 200-300$) but could not refine within that band, making the guidance ineffective. If the guidance itself is not converged, our method also does not make a difference. We attribute this to (i) noisy returns and (ii) long-horizon credit assignment, both exacerbated in offline settings without on-policy rollouts. By contrast, TD-based (s, a)-level guidance provides stable, local gradients that are compatible with diffusion updates.

E Experimental Setup and Hyperparameters

.All models are trained using the D4RLMuJoCoTD Dataset Fu et al. (2021). It provides pre-collected trajectories of varying quality, including expert, medium-expert, medium-replay, and medium datasets, enabling rigorous training of offline RL algorithms under diverse data distributions. The evaluation is done in randomly initialized environments. All the gradient steps mentioned are with respect to a batch size of 256.

Wherever possible, we adopt the original hyperparameter settings from the paper of Wang et al. (2024). Intentionally, we do not modify any training-related components—including the optimizer, learning rate, batch size, architecture, or loss function. Because a key strength of our method is that it achieves superior performance without requiring any changes to the other parameters other than pretraining or modularize. This highlights the robustness and plug-and-play nature of our approach. Final results are reported, averaged over 50 evaluation episodes. Performance is measured by normalized return, and we report both the mean(Section 4.1) and variance deduction(Section **??**).

All experiments were implemented in PyTorch and based on the CleanDiffuser framework Wang et al. (2024). We strictly followed the hyperparameter settings from the baseline implementations to ensure fair comparison. Table 9 summarizes the key values.

Table 9: Hyperparameters used in our experiments (inherited from CleanDiffuser).

Parameter	**Value**	**Notes**
Optimizer	Adam	
Learning Rate	3×10^{-4}	fixed across all models
Batch Size	256	
Discount Factor γ	0.99	
Noise Schedule	cosine	unless otherwise specified
Number of Diffusion Steps T	5,10,20,30,40	
Actor Loss Weight η	1.0	scales $\mathcal{L}_Q$

For reproducibility, we will release full training scripts and environment configurations in our code repository.

F Why the generated action with the GFDT and the modular methods are in distribution

Addressing a Key Concern One might naturally worry that even if the guidance module (e.g., the Q-function Q_ϕ) and the diffusion model are both trained solely on the offline dataset $\mathcal{B}$, their combination during sampling could still produce out-of-distribution actions. The value guidance term $\nabla_a Q_\phi(a)$ may have a large magnitude or steep gradients, which could push the sampled action a_t away from the data manifold if not properly controlled. Therefore, the perturbation magnitude and the guidance magnitude are tightly controlled, and the guidance gradient is normalized :

$$\|a_{t+1} - a_t\| \leq \lambda + \mathcal{O}(\sqrt{\tau_t}), \lambda > 0 \ and \ \lambda \to 0$$

ensuring that each sampling step stays within a small neighborhood of the current point. Since the diffusion model is trained on the dataset $\mathcal{B}$ and generates samples close to it, and since the guidance is applied as a *soft* correction, the overall sampling trajectory remains near the support of $\mathcal{B}$. Thus, even when guided by Q_ϕ, the diffusion process remains effectively batch-constrained.

G Comparison Table of GFDT

This section contains two performance comparison tables Table.10 and Table. 11, that shows the percentages of improvement of GDFT and as other methods, compared to baseline models.

Table 10: Mujoco and Antmaze results of DQL. Each cell shows the raw score and the relative performance (%).

Environment	Baseline	GFDT	GFDT(%)	GAI	GAI(%)	BC	BC(%)	Unfreeze(%)
halfcheetah-expert	88.21	90.43	102.51%	68.49	77.65%	85.82	97.29%	101.47%
halfcheetah-medium-expert	88.28	90.18	102.15%	89.73	101.64%	85.53	96.88%	101.67%
halfcheetah-medium-replay	67.38	67.93	100.81%	67.76	100.57%	55.87	82.92%	100.08%
halfcheetah-medium	59.88	66.99	111.88%	53.36	89.12%	54.57	91.13%	91.93%
hopper-expert	166.76	172.90	103.68%	162.68	97.55%	165.11	99.01%	93.03%
hopper-medium-expert	168.00	172.31	102.56%	166.28	98.98%	166.88	99.33%	94.70%
hopper-medium-replay	151.59	152.98	100.91%	121.97	80.46%	113.30	74.74%	99.28%
hopper-medium	143.92	147.06	102.18%	71.09	49.40%	118.08	82.04%	100.47%
walker2d-expert	117.25	120.25	102.56%	119.91	102.28%	108.05	92.16%	100.57%
walker2d-medium-expert	117.73	117.63	99.91%	115.08	97.75%	117.45	99.76%	100.65%
walker2d-medium-replay	92.96	95.62	102.86%	80.32	86.40%	87.14	93.74%	102.86%
walker2d-medium	87.65	87.89	100.27%	77.33	88.22%	82.28	93.87%	100.27%
Average	112.47	115.18	102.69%	99.50	89.17%	103.34	91.91%	98.91%
antmaze-large-diverse-v2	63.33	90.67	143.16%	86.67	136.84%	66.00	104.21%	136.84%
antmaze-large-play-v2	90.00	89.33	99.26%	88.67	98.52%	50.67	56.30%	101.48%
antmaze-medium-diverse-v2	93.33	97.33	104.29%	94.00	100.71%	78.00	83.57%	102.86%
antmaze-medium-play-v2	67.33	91.33	135.64%	88.67	131.68%	68.00	100.99%	138.61%
Average	78.50	92.17	120.59%	89.50	116.94%	65.67	86.27%	119.95%

Table 11: Mujoco and Antmaze results of EDP. Each cell shows the raw score and the relative performance (%).

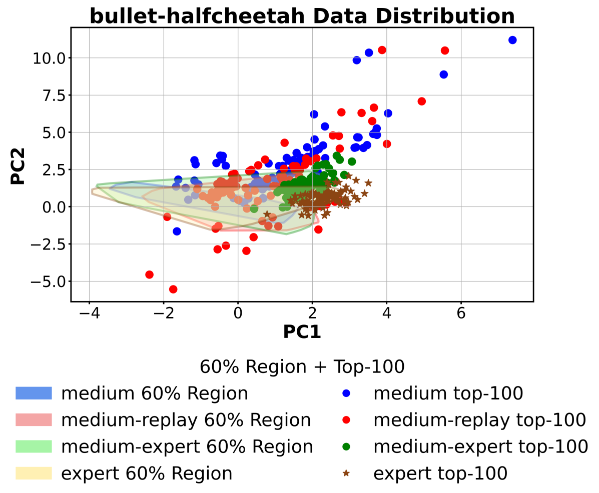
Environment	EDP_baseline	GFDT	GFDT(%)	GAI	GAI(%)	BC	BC(%)	Unfreeze(%)
halfcheetah-expert	86.55	86.82	100.30%	78.78	91.02%	85.82	99.15%	99.34%
halfcheetah-medium-expert	86.74	87.16	100.48%	78.18	90.14%	85.53	98.61%	100.12%
halfcheetah-medium-replay	65.78	64.42	97.94%	47.08	71.57%	55.87	84.94%	97.94%
halfcheetah-medium	54.59	59.26	108.57%	53.56	98.12%	54.57	99.96%	101.36%
hopper-expert	161.23	163.65	101.50%	151.93	94.23%	165.11	102.41%	100.63%
hopper-medium-expert	161.36	163.95	101.61%	141.07	87.43%	166.88	103.42%	101.44%
hopper-medium-replay	112.16	139.75	124.60%	114.32	101.93%	113.30	101.02%	106.91%
hopper-medium	141.16	141.64	100.35%	83.25	58.98%	118.08	83.65%	100.27%
walker2d-expert	116.14	116.34	100.18%	114.27	98.39%	108.05	93.04%	100.19%
walker2d-medium-expert	116.31	116.59	100.24%	109.55	94.19%	117.45	100.98%	100.65%
walker2d-medium-replay	85.02	83.77	98.53%	68.84	80.97%	87.14	102.49%	100.19%
walker2d-medium	84.24	84.21	99.96%	58.21	69.10%	82.28	97.67%	100.24%
Average	105.94	108.96	102.86%	91.59	86.34%	103.34	97.28%	100.77%
antmaze-large-diverse-v2	30.67	34.67	113.05%	28.67	93.48%	10.67	34.78%	154.35%
antmaze-large-play-v2	21.33	22.67	106.25%	18.67	87.50%	18.67	87.50%	118.75%
antmaze-medium-diverse-v2	67.33	52.00	77.23%	2.00	2.97%	18.00	26.73%	19.80%
antmaze-medium-play-v2	73.30	118.00	160.98%	90.00	122.78%	76.00	103.68%	154.62%
Average	48.16	56.83	114.38%	34.83	76.68%	30.83	63.17%	111.88%

H Detailed Analysis of Plug-and-play

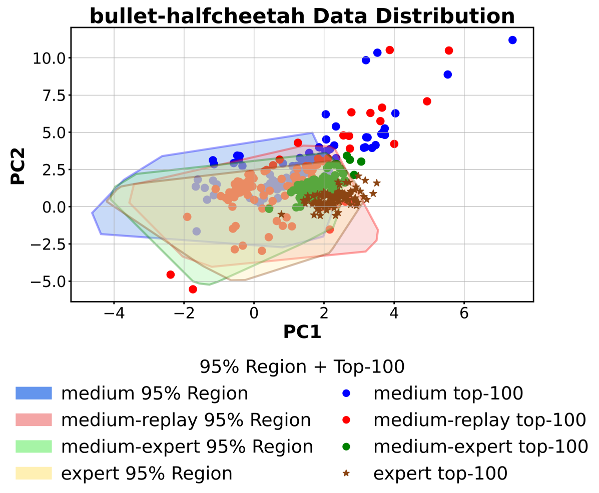
	DGID: DGID (DQL-Guidance + IDQL-Diffusion): DGID follows the overall pipeline of IDQL, meaning that its generation and diffusion process are still driven by IDQL's methodology. The key difference is that the component responsible for selecting optimal action candidates has been replaced by a DQL module.	**IGDD: IGDD (IDQL-Guidance + DQL-Diffusion):** IGDD is built upon DQL's architecture and training logic, but replaces its gradient-based optimization module with the advantage-guided update mechanism from IDQL.
MUJOCO Environment Characteristics: MuJoCo's dense-reward nature provides immediate evaluative feedback at every timestep. This continuous supervision allows the agent to quickly identify and correct suboptimal actions, effectively smoothing the learning curve. Consequently, the environment is forgiving to small deviations or approximation errors, as incorrect behaviors are rapidly penalized and adjusted.	Compared to the original IDQL, DGID achieves a significant performance improvement, with an average score of 112.75, and becomes competitive with the baseline DQL model. The DQL module offers a strong optimization signal that effectively give IDQL a most deterministically optimal guidance on the action choice, during inference. Since the MuJoCo environment provides dense and continuous rewards, the system can directly benefit from DQL's precise action evaluation. The dense-reward setting also makes the model more tolerant to small distribution mismatches between DQL and IDQL(errors corrected immediately), leading to stable and superior overall performance.	This effectively smooths the optimization trajectory and regularizes the learning process. IGDD performs slightly better than the original DQL, achieving an average score of 115.97 and showing improved stability during diffusion. The IDQL-style advantage guidance provides smoother gradients and prevents overly aggressive Q-value maximization, which is a known issue in pure DQL setups. The result is a more balanced and robust optimization process that preserves DQL's iterative refinement strength while improving convergence reliability. In dense-reward environments like MuJoCo, where feedback is immediate and continuous, this regularized update leads to steady and consistent performance gains.
antmaze: the task follows a typical sparse-reward setting, where the agent only receives a positive signal upon successful completion of the maze. Due to the extremely long horizons—often spanning thousands of steps—the model requires high precision and the accumulated errors are not easily corrected. Moreover, the environment is heavily affected by noise and stochastic dynamics, making it highly sensitive to approximation errors.	It is suspected that When the DQL guidance is applied to out-of-distribution actions generated by IDQL, which does not generate the most rewarding signal every step, its Q-values become unreliable and often misleadingly high.(recall that mujoco is less noisy and easy to correct errors). The sharp, deterministic MaxQ operator further amplifies these estimation errors, leading to unstable and degraded performance. The main reason for the lower performance of PAP is antmaze needs conservative high precision and the broken of assumption of "small enough steps and long enough trajectory" became several in this sensitive environment.	(DGID follows IDQL's pipeline, but replaces its action selection with DQL's MaxQ-guided module). IGDD achieves a suboptimal result in AntMaze, with an average score of 62.17. It underperforms relative to original DQL but remains noticeably better than DGID. Because the IDQL-style guidance provides weaker but smoother signals, it is insufficient for the sparse-reward nature of AntMaze. This environment only provides a final success reward, so precise long-horizon navigation depends on sharp gradient signals to correctly guide early steps. The lack of strong feedback causes gradual error accumulation across iterations, leading to less efficient exploration and reduced success rates.

I Details of Application Scope

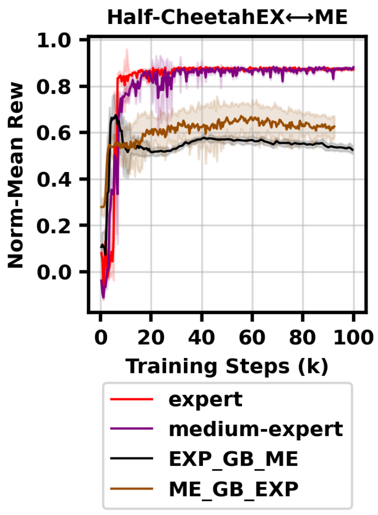
These are all the plots of cross experiments. We did not perform data mixing experiments because, as shown in the study by Miao et al. (2023), mixing datasets Generated from different policies can lead to degraded performance. Our ablation on the OOD dataset does not involve dataset mixing, since the diffusion model and the guidance model are each trained on only one dataset. Consequently, the diffusion model does not learn from multiple policies. However, when the guidance model is trained on a different dataset, it may fail to provide correct gradient signals.



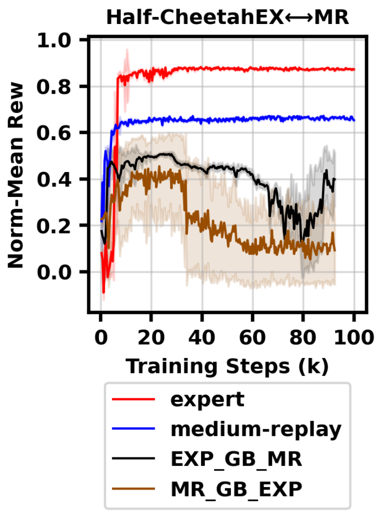
(a) density hulls (top 100, $k = 20$) of Half Cheetah



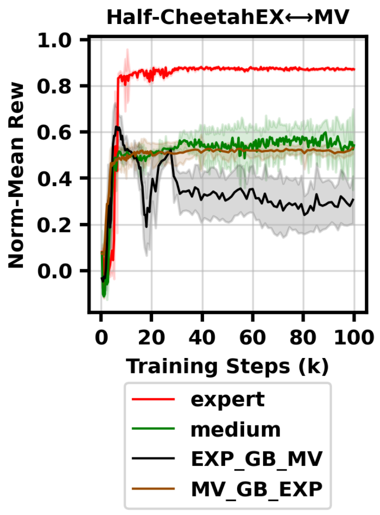
(b) PCA hull and top 100 samples of Half Cheetah



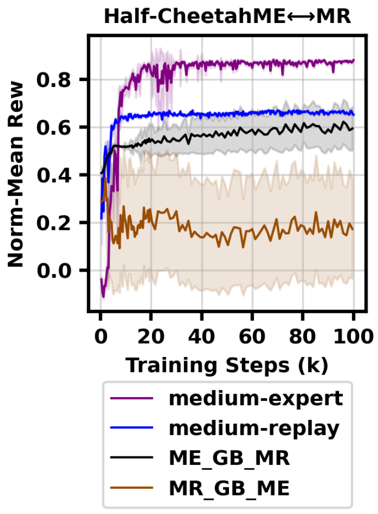
(a) Aross Experiments of Expert(EX) and Medium Export(ME)



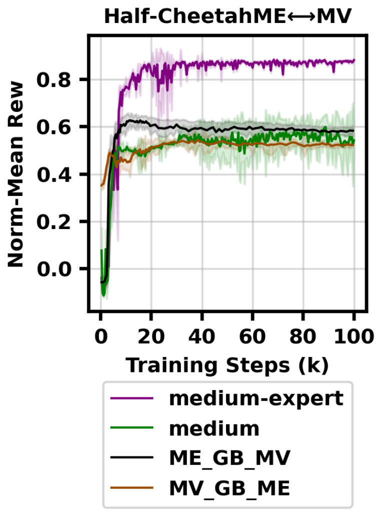
(b) Aross Experiments of Expert(EX) and Medium Replay(MR)



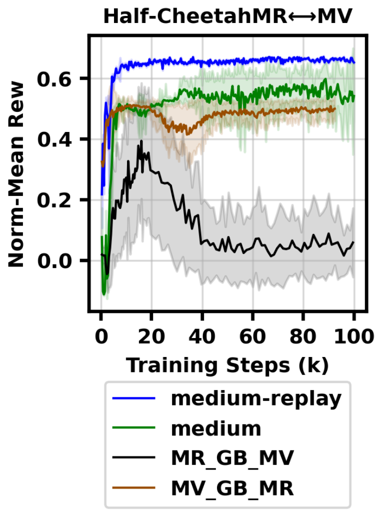
(c) Aross Experiments of Expert(EX) and Medium(MV)



(d) Aross Experiments of Medium Expert(ME) and Medium Replay(MR)

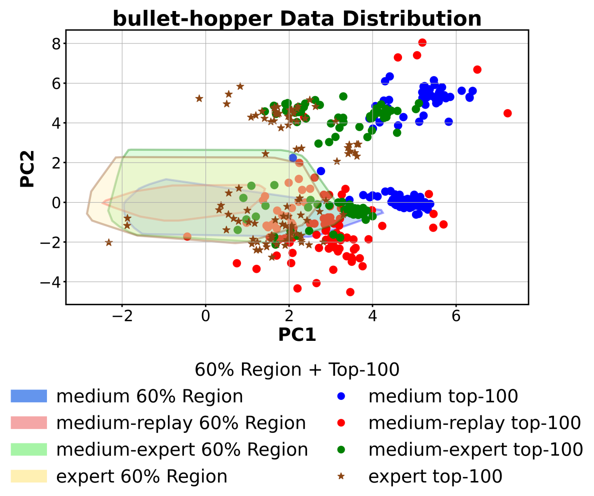


(e) Aross Experiments of Medium Expert(ME) and Medium(MV)

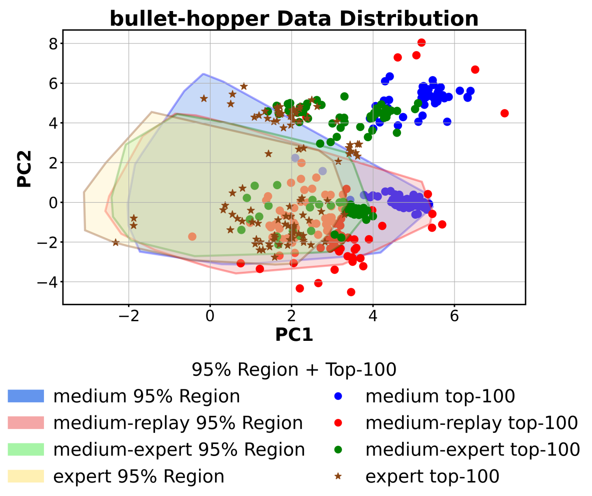


(f) Aross Experiments of Medium Replay(MR) and Medium(MV)

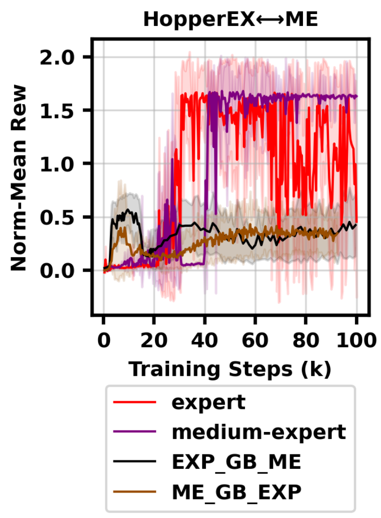
Figure 9: Half-cheetah Dataset Experiments



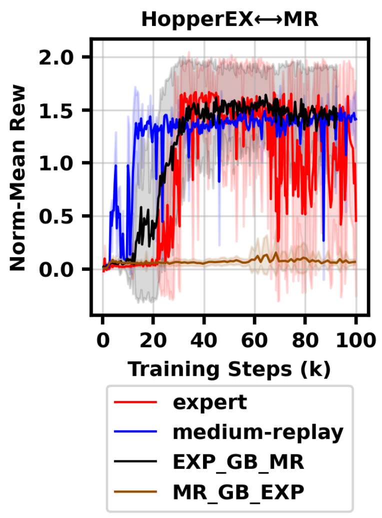
(a) density hulls (top 100, $k = 20$) of Hopper



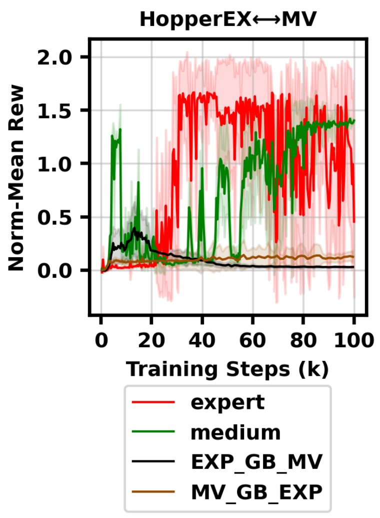
(b) PCA hull and top 100 samples of hopper



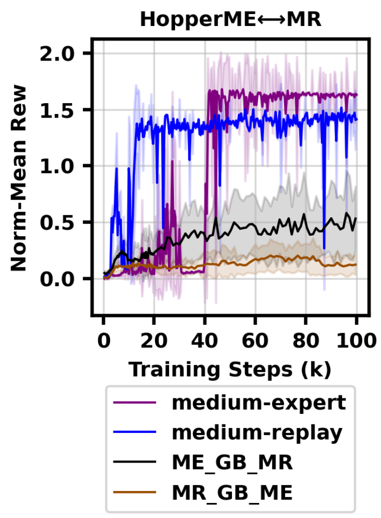
(a) Aross Experiments of Expert(EX) and Medium Expert(ME)



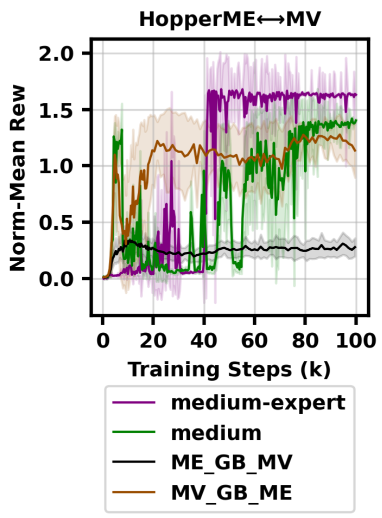
(b) Aross Experiments of Expert(EX) and Medium Replay(MR)



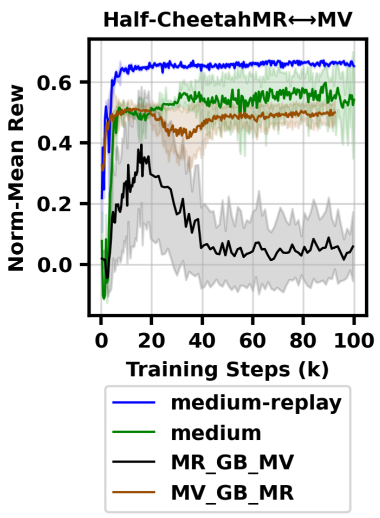
(c) Aross Experiments of Expert(EX) and Medium(MV)



(d) Aross Experiments of Medium Expert(ME) and Medium Replay(MR)

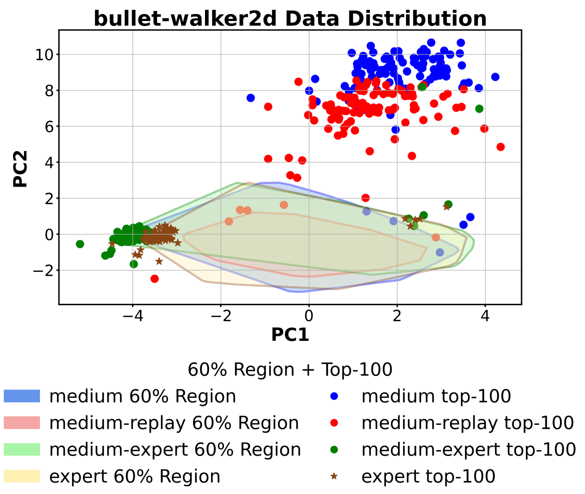


(e) Aross Experiments of Medium Expert(ME) and Medium(MV)

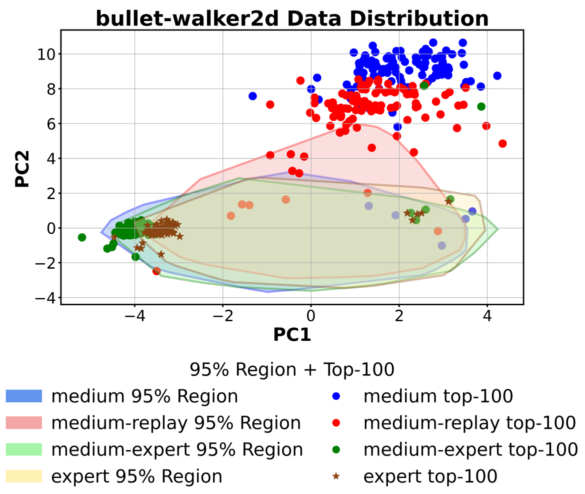


(f) Aross Experiments of Medium Medium Replay(MR) and Medium(MV)

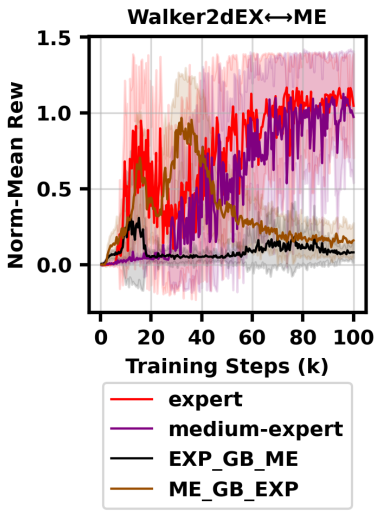
Figure 11: Hopper Dataset Experiments



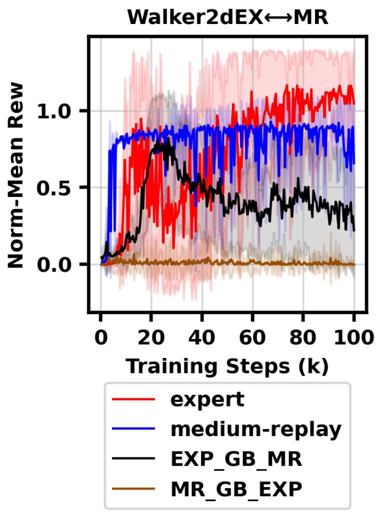
(a) density hulls (top 100, $k = 20$) of Walker2d



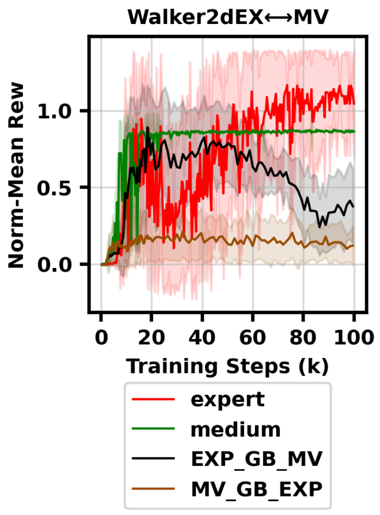
(b) PCA hull and top 100 samples of Walker2d



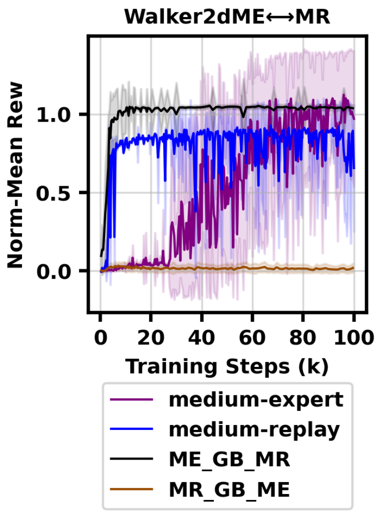
(a) Aross Experiments of Expert(EX) and Medium Expert(ME)



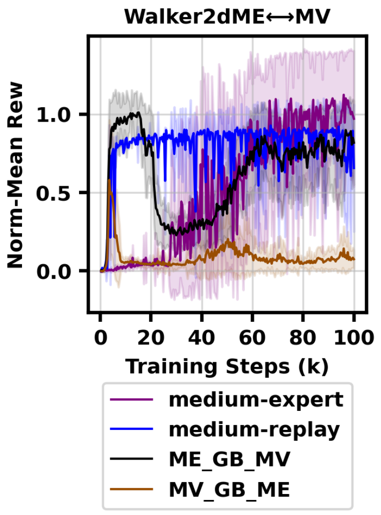
(b) Aross Experiments of Medium Expert(EX) and Medium Replay(MR)



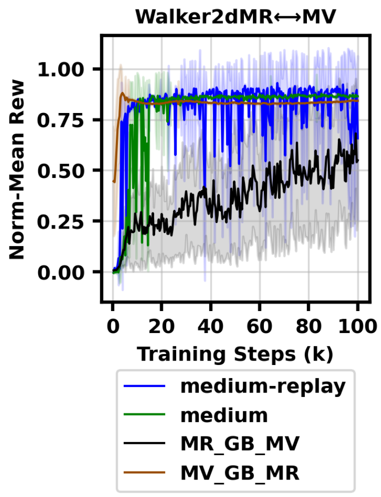
(c) Aross Experiments of Medium Expert(EX) and Medium(MV)



(d) Aross Experiments of Medium Expert(ME) and Medium Replay(MR)



(e) Aross Experiments of Medium Expert(ME) and Medium(MV)



(f) Aross Experiments of Medium Replay(MR) and Medium(MV)

J Notation and Terminology

Table 12: Equation Terminologies

Symbol	**Type**	**Meaning**
$\mathcal{D}$	set	Offline dataset of transitions or (s, a_0) pairs
s, a, r, s'	var	State, action, reward, and next state
$\pi_\theta(a \mid s)$	policy	Diffusion policy with parameters θ
$\epsilon_\theta(s, x_t, t)$	net	Noise predictor (denoiser)
$Q_\phi(s, a)$	net	Q-function with parameters ϕ
a_0	action	Ground-truth clean action at $t = 0$
$\hat{a}_0$	action	Reconstructed estimate of a_0 via ϵ_θ
x_t	action	Noisy action at diffusion step t
t, T	index	Diffusion timestep and total steps
α_t, $\bar{\alpha}_t$	sched	Per-step and cumulative noise schedule coefficients
ϵ	noise	Standard Gaussian noise
η_g	scalar	Step size for one-step Q-guidance (EDP)
λ	scalar	Guidance scale during inference
L_{BC}	loss	Behavior cloning loss, $\|\epsilon - \hat{\epsilon}\|^2$
L_Q	loss	Q-guided loss term
L_{actor}	loss	Actor loss, e.g. $L_{\text{BC}} + \eta\, L_Q$
γ	scalar	Discount factor in TD targets
K	count	Number of candidate actions sampled (IDQL inference)

Table 13: Symbol definitions and meanings

Table 14: Algorithm Abbreviations

Abbreviation	**Explanation**
Env	**Environment.** The benchmark task or environment in which the algorithm is evaluated (e.g., HalfCheetah, Hopper, Walker2d, AntMaze).
ReBR	**Regularized Behavior Regularized Actor Critic (ReBRAC).** A conservative offline RL algorithm emphasizing stability through behavior regularization (Wu et al., 2019).
DICE	**Dynamic Importance Sampling Correction Estimator.** An offline RL method that uses importance weighting to correct distribution mismatch during policy evaluation (Ma et al., 2024).
CQL	**Conservative Q-Learning.** A value-based offline RL algorithm that penalizes overestimation of unseen actions to ensure conservative value estimation (Kumar et al., 2020).
IQL	**Implicit Q-Learning.** A decoupled offline RL method that learns Q-values and implicitly defines a policy via advantage-weighted regression without explicit policy optimization (Kostrikov et al., 2022).
DQL_GF	**Diffusion Q-Learning with Guidance-First (GFDT).** The DQL algorithm trained under the Guidance-First Diffusion Training paradigm, where the Q-network is pretrained and frozen before diffusion training.
DQL_D_GF	**Double-Guidance Diffusion Q-Learning.** A DQL-GFDT variant that replaces the guidance module at inference with an independently initialized version (different random seed) to reduce variance.
EDP_GF	**Efficient Diffusion Policy with Guidance-First (GFDT).** A one-step denoising diffusion policy algorithm enhanced with pretrained frozen guidance to accelerate convergence and improve efficiency.
EDP_D_GF	**Double-Guidance Efficient Diffusion Policy.** An EDP-GFDT variant employing an independently initialized second guidance network at inference for improved stability and reduced variance.
GFDT	**Guidance-First Diffusion Training.** A training paradigm where the guidance (Q-network) is pretrained and frozen before training the diffusion policy.
Baseline	The original implementation of the corresponding diffusion-based offline RL algorithm (e.g., DQL, IDQL, or EDP) without our proposed modifications.
D-Baseline	**Double-Guidance Baseline.** A variant of the baseline model where the guidance module used during inference is replaced with an independently initialized version of the same architecture (different random seed).
Double_GFDT	GFDT equipped with *Double Guidance* at inference time, reducing variance through independent initialization.
GAI	**Guidance at Inference.** A behavior cloning model trained without reward guidance during training, but augmented with Q-guidance only at inference.
Unfreeze	A variant of GFDT where the pretrained guidance module is not frozen but continues to be updated during policy training.
BC	**Behavior Cloning.** A supervised learning baseline that trains a policy to imitate dataset actions without reward guidance.

Table 15: D4RL Benchmark Environment Abbreviations

Abbreviation	**Explanation**
HCEX	**HalfCheetah-Expert.** Dataset generated by an expert policy in the HalfCheetah environment.
HCME	**HalfCheetah-Medium-Expert.** Dataset generated by a mixture of medium and expert policies in HalfCheetah.
HCMR	**HalfCheetah-Medium-Replay.** Dataset generated by replay buffer data collected from medium-performance policies.
HCMV	**HalfCheetah-Medium.** Dataset generated by a medium-performance policy in HalfCheetah.
HOEX	**Hopper-Expert.** Dataset generated by an expert policy in the Hopper environment.
HOME	**Hopper-Medium-Expert.** Dataset generated by a mixture of medium and expert policies in Hopper.
HOMR	**Hopper-Medium-Replay.** Replay buffer dataset in Hopper.
HOMV	**Hopper-Medium.** Dataset generated by a medium policy in Hopper.
WAEX	**Walker2d-Expert.** Expert policy dataset in Walker2d.
WAME	**Walker2d-Medium-Expert.** Mixed dataset in Walker2d.
WAMR	**Walker2d-Medium-Replay.** Replay dataset in Walker2d.
WAMV	**Walker2d-Medium.** Medium policy dataset in Walker2d.

References

Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B. Tenenbaum, Tommi S. Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision making? In *International Conference on Learning Representations*, 2023. Top 5% of submissions.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, 2017. URL `https://arxiv.org/abs/1704.06643`.

Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. In *Advances in Neural Information Processing Systems*, 2021.

The Farama Foundation. Gymnasium: A standard api for reinforcement learning environments. https://github.com/Farama-Foundation/Gymnasium, 2024.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2021. URL `https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/file/7b5b23f4aadf9513306bcd59afb6e4c9-Paper-round2.pdf`.

Scott Fujimoto, Edoardo Conti, Mohammad Ghavamzadeh, and Joelle Pineau. Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint arXiv:1910.01708*, 2019a.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pp. 2052–2062, 2019b.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. In *International Conference on Machine Learning*, pp. 3929–3938, 2020.

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*, volume 33, pp. 6840–6851, 2020.

Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In *Conference on Empirical Methods in Natural Language Processing*, pp. 8741–8754, 2021.

Gautier Izacard, Edouard Grave, Fabio Petroni, Lucas Hosseini, Timo Schick, Pierre-Emmanuel Mazare, Julian Eisenschlos, Sebastian Petrov, Vladimir Karpukhin, Patrick Lewis, Wen-tau Yih, Tim Rocktaschel, Sebastian Riedel, and Douwe Kiela. Atlas: Few-shot learning with retrieval augmented language models. In *International Conference on Machine Learning*, 2022.

Michael Janner, Yilun Du, Joshua B. Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, 2022. URL `https://arxiv.org/abs/2205.09991`.

Bingyi Kang, Haotian Shi, Yilun Tan, Mingming Zhu, Dahuan Lin, and Hang Zhou. Efficient diffusion policies for offline reinforcement learning. *arXiv preprint arXiv:2310.03573*, 2023.

Dongjun Kim, Yeongmin Kim, Se Jung Kwon, Wanmo Kang, and Il-Chul Moon. Refining generative process with discriminator guidance in score-based diffusion models. In *International Conference on Machine Learning*, volume 202, pp. 1–25. PMLR, 2023.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations (ICLR)*, 2022.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1179–1191, 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.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, 2020.

Cheng Lu, Shengming Ruan, Yang Song, and Stefano Ermon. Energy-based guidance for diffusion models. In *International Conference on Machine Learning*, 2023.

Xingyu Ma, Yuchen Yang, Yifei Chen, Han Liu, and Tong Zhang. Dice: Offline reinforcement learning with diffusion models. In *International Conference on Learning Representations (ICLR)*, 2024.

Xiyang Ma, Fangchen Luo, Nan Jiang, Ofir Nachum, and Dale Schuurmans. Revisiting the minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/hash/f0e5d0b6e6bfb14d91d60a3f3e4f0c76-Abstract-Conference.html`. NeurIPS 2023 Conference Paper.

Q. Miao et al. Learning with noisy labels using collaborative sample selection. In *Advances in Neural Information Processing Systems*, 2023.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL `https://proceedings.neurips.cc/paper/2018/file/1802.05957-Paper.pdf`.

Chong Mou, Yabo Zhang, Yixiao Li, et al. T2i-adapter: Learning adapters to dig out more controllable ability for text-to-image diffusion models. *arXiv preprint arXiv:2302.08453*, 2023.

Ashvin Nair, Aviral Kumar, Chelsea Finn, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. In *Advances in Neural Information Processing Systems*, 2020.

Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pam Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. In *Advances in Neural Information Processing Systems*, 2021.

Xue Bin Peng, Angjoo Kanazawa, Sam Toyer, Pieter Abbeel, and Sergey Levine. Composing complex skills by learning transition policies. In *IEEE International Conference on Robotics and Automation*, 2019. URL `https://arxiv.org/abs/1906.01068`.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, 2021.

Hado van Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, volume 23, pp. 2613–2621, 2010.

Zhendong Wang, Xinyang Zhang, Bowen Liu, et al. Cleandiffuser: Diffusion library and benchmark for decision-making. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL `https://arxiv.org/abs/2406.09509`.

Zhendong Wang et al. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2023.

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 7968–7978, 2019.

Zexiang Ye, Yixiao Zhang, et al. Ip-adapter: Text-compatible image prompt adapter for text-to-image diffusion models. *arXiv preprint arXiv:2308.06721*, 2023.

Tianhe Yu, Aviral Kumar, Yevgen Chebotar, Karol Hausman, Gaurav Sukhatme, Sergey Levine, and Chelsea Finn. Mopo: Model-based offline policy optimization. In *Advances in Neural Information Processing Systems*, pp. 14129–14142, 2020.

Lvmin Zhang, Yongjie Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *IEEE International Conference on Computer Vision*, 2023.