PADGEN: POLICY-ADAPTIVE DATA GENERATION FOR OFFLINE REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Data augmentation has been widely used to improve offline reinforcement learning (RL) by expanding the available training data. However, existing methods largely follow a static generation paradigm, where augmented data remain fixed throughout policy optimization, overlooking that *data utility changes as the policy evolves*. Consequently, statically generated data may become less useful as training progresses and fail to accommodate the evolving data needs of the policy, thereby limiting further policy improvement. To address this limitation,we propose a policy-adaptive data generation framework (PADGen), which consists of a generator, selector, and policy. The generator generates data based on the utility feedback from both the selector and the policy. The selector evaluates and selects generated data according to their utility for policy optimization. After each training iteration, the policy provides optimization feedback, while the selector returns the utility evaluation of the generated data. The generator then leverages these feedback signals to adaptively generate new data for the next iteration, forming a closed-loop data generation and policy optimization process. This design enables the generated data to be continuously adjusted according to the current policy, improving their utility throughout policy optimization. Extensive experiments across multiple offline RL algorithms and benchmark datasets demonstrate that PADGen consistently outperforms existing data augmentation methods. Our code is available at `https://anonymous.4open.science/r/PADGen-ABCD`.

1 INTRODUCTION

Offline reinforcement learning (Offline RL) (Levine et al., 2020; Fu et al., 2020; Fujimoto et al., 2019; Kumar et al., 2019; 2020; Janner et al., 2022; Ajay et al., 2023; Rafailov et al., 2024; Nishimori et al., 2025; Fellows et al., 2026) aims to learn effective policies solely from a pre-collected dataset, without further interaction with the environment. By decoupling policy learning from online environment interaction, Offline RL provides an appealing paradigm for scenarios where data collection is consuming and risky, such as robotics (Kumar et al., 2022; Rafailov et al., 2024; Corrado et al., 2024; Nishimori et al., 2025), autonomous driving (Shi et al., 2021; Lee et al., 2024a), and healthcare (Fatemi et al., 2022; Kausik et al., 2024; Zhou et al., 2024). However, this advantage also imposes a fundamental constraint: The policy can only learn from the information contained in the available offline dataset and lacks a mechanism to actively acquire additional data when the dataset is insufficient.

To alleviate this limitation, recent studies (Sinha et al., 2022; Lu et al., 2023; Lee et al., 2024b; Li et al., 2024; Kidambi et al., 2020; Wang et al., 2021) have explored static data augmentation for Offline RL as illustrated in Figure 1(c), aiming to enrich the training data beyond the original offline dataset. Existing methods typically augment the dataset by modeling the environment dynamics and generating synthetic rollouts (Yu et al., 2020; Zhang et al., 2023; Kidambi et al., 2020; Wang et al., 2021; Lyu et al., 2022), perturbing states or transitions (Sinha et al., 2022; Lu et al., 2023), or generating and stitching trajectory segments (Lee et al., 2024b; Li et al., 2024; Wang et al., 2026). The resulting generated samples can be incorporated into policy optimization to enlarge the training set while remaining fixed throughout subsequent training iterations.

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.

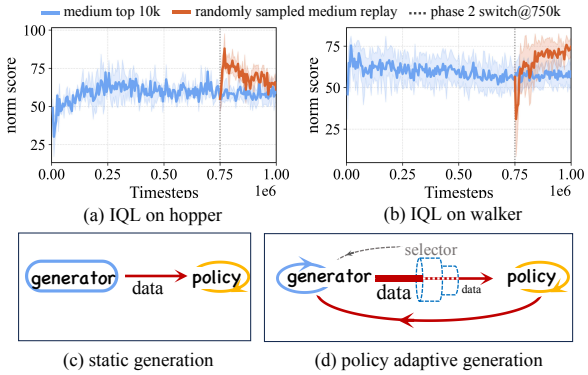


Figure 1: Motivation of policy-adaptive generation. (a) and (b) The dynamic utility of data during policy optimization, where introducing randomly collected data leads to further performance improvements. (c) and (d) Comparison between the conventional static generation paradigm and our policy-adaptive generation paradigm.

Despite their effectiveness, such a static paradigm overlooks that the utility of data may change as the policy evolves (Schaul et al., 2016; Schweighofer et al., 2022). As it is illustrated in Figure 1(a) and (b), we first train IQL on the 10K highest-return transitions from the Medium dataset for 750K steps. We then continue training for another 250K steps under two data settings: (1) continuing with the same 10K high-return transitions used in the first stage; and (2) replacing them with randomly sampled 10K Medium-Replay transitions, which have slightly lower average returns than the original training data. The results shown that continued training on the original high-return data leads to performance degradation, whereas introducing randomly sampled data improves performance. These results suggest that the utility of data changes throughout policy optimization: data that improves the policy at one stage may no longer provide effective learning signals at later stages. Consequently, instead of repeatedly training policies on statically generated data, data generation should adapt to the evolving needs of policy learning.

This observation motivates a shift from conventional static data augmentation toward policy-adaptive data generation, where the generation process is dynamically guided by feedback from the evolving policy optimization. To this end, we propose a policy-adaptive data generation framework (PADGen), as shown in Figure 1 (d). Our PADGen consists of three key components: a generator, a selector, and a policy. The generator generates new data by leveraging utility feedback provided by both the selector and the policy. The selector assesses the generated samples and identifies those with higher quality for policy optimization. During each training iteration, the policy returns optimization signals reflecting its current learning status, while the selector provides quality feedback on the generated data. Based on these two complementary feedback signals, the generator updates its generation strategy for subsequent iterations, thereby establishing a closed-loop interaction between data generation and policy optimization. To validate the effectiveness of PADGen, we conduct extensive experiments on widely used offline RL benchmarks with representative offline RL algorithms. The experimental results demonstrate that PADGen consistently improves policy performance over existing data augmentation methods. Furthermore, we observe that PADGen is particularly effective when the underlying offline datasets contain substantial low-quality data.

In summary, the contributions of this paper are:

- We propose PADGen, a policy-adaptive data generation framework that establishes a closed-loop interaction between data generation and policy optimization.
- To the best of our knowledge, PADGen is the first framework to adapt the data generation process according to the evolving policy during training.
- Extensive experiments on the widely used benchmarks and representative offline RL algorithms demonstrate the effectiveness of PADGen.

2 Related Works

To alleviate the limitation that Offline RL cannot collect additional data through environment interaction, existing studies (Sinha et al., 2022; Lu et al., 2023; Lee et al., 2024b; Huang et al., 2024; Qing et al., 2025; Schmähling et al., 2026; Li et al., 2024; Lee & Choi, 2026; Wang et al., 2026; Yu et al., 2020; 2021; Corrado et al., 2024) mainly resort to data augmentation to enrich the fixed offline datasets. Early approaches focus on state- or transition-level augmentation, such as S4RL (Sinha et al., 2022), which perturbs states to improve value function generalization, and SynthER (Lu et al., 2023), which uses diffusion models to generate synthetic transitions. Recent studies further extend augmentation to the trajectory level. GTA (Lee et al., 2024b) and GODA (Huang et al., 2024) generate trajectories conditioned on return or goals, while BiTrajDiff (Qing et al., 2025) and

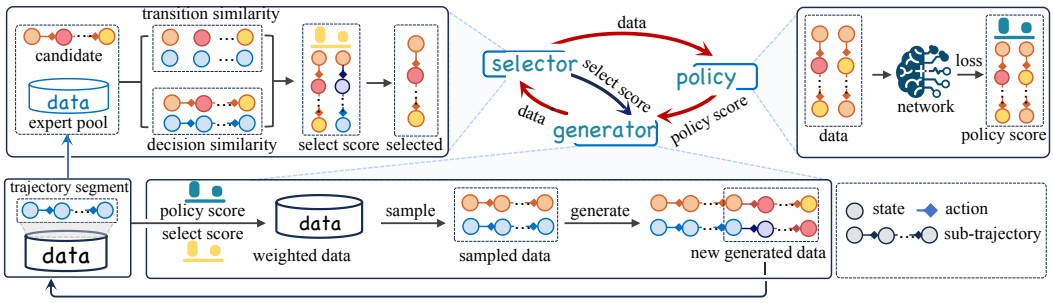


Figure 2: PADGen consists of three components: a generator, a selector, and a policy. The generator produces augmented data conditioned on the selection and policy scores. The selector evaluates the generated data to obtain selection scores for data selection, while the policy is trained on the selected data and provides policy scores reflecting its learning status. Both scores are fed back to the generator to guide subsequent data generation, forming a closed-loop augmentation process.

LIFT(Schmähling et al., 2026) improve trajectory diversity by exploiting the structure of offline trajectories. Another line of work explores trajectory stitching to expand the coverage and quality of offline data, including DiffStitch (Li et al., 2024) and SCoTS (Lee & Choi, 2026). TGCVG (Wang et al., 2026) further incorporates conservative value guidance into trajectory generation to produce high-quality trajectories while controlling distribution shift. Despite these advances, existing methods primarily focus on static data generation, in which the generated data remains fixed throughout policy training iterations. However, the utility of generated data can change with the evolving policy, making data that is useful at one stage of training less informative at another. This motivates a shift from conventional static data augmentation toward policy-adaptive data generation, where the generation process is guided by the current needs of the policy optimization.

3 Preliminaries

Offline Reinforcement Learning We consider a Markov Decision Process (MDP) defined as $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \rho_0, p, r, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, ρ_0 is the initial state distribution, $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ represents the transition dynamics, $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. At each time step t, an agent observes a state $\boldsymbol{s}_t \in \mathcal{S}$ and takes an action $\boldsymbol{a}_t \sim \pi(\cdot|\boldsymbol{s}_t)$ according to policy π. The environment then returns a reward r_t and transits to the next state $\boldsymbol{s}_{t+1}$ according to $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$. The resulting interaction forms a trajectory $\tau = \{(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t)\}_{t=0}^{T}$. The objective of reinforcement learning is to learn a policy that maximizes the expected discounted return (Sutton & Barto, 2018) $J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$.

In offline reinforcement learning (offline RL), the policy is learned from a fixed dataset of previously collected trajectories, without additional interaction with the environment during training (Levine et al., 2020; Kumar et al., 2022). Let $\mathcal{D} = \{\tau_i\}_{i=1}^{N}$ denote the offline dataset. Given an offline RL algorithm Γ, the learned policy is $\pi_{\mathcal{D}} = \Gamma(\mathcal{D})$, and its performance can be evaluated by its expected return in the environment $J(\Gamma(\mathcal{D})) = \mathbb{E}_{\tau\sim\pi_{\mathcal{D}}}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$.

Data Augmentation for Offline RL The performance of offline RL is closely related to the information contained in the training dataset (Levine et al., 2020; Schweighofer et al., 2022). Data augmentation provides additional training samples by generating new data from an existing offline dataset. Given an offline dataset $\mathcal{D}$, the data augmentation could be described by $\mathcal{D}^{\text{aug}} = \mathcal{G}(\mathcal{D})$, where $\mathcal{D}^{\text{aug}}$ denotes the additional data generated from or conditioned on the original dataset. The augmented dataset can then be used together with the original data for policy learning $\pi_{\mathcal{D}^+} = \Gamma\left(\mathcal{D} \cup \mathcal{D}^{\text{aug}}\right)$. Data augmentation therefore provides a way to enrich the available training data without requiring additional interaction with the environment.

4 Methodology

Unlike the existing static generation paradigm, PADGen adopts a policy-adaptive generation paradigm, where data generation dynamically adapts to the evolving needs of the policy (Schaul et al., 2016; Jiang et al., 2021). As illustrated in Figure 2, PADGen consists of three components: a generator, a selector, and a policy. The generator, trained on the offline dataset, generates new data by leveraging two complementary feedback signals: optimization feedback from the policy and

quality feedback from the selector. The generated data are then passed to the selector for evaluation and selection. The selector provides informative samples to the policy for optimization while returning quality feedback to the generator to guide subsequent data generation. The policy is optimized on the selected samples and, in turn, provides training signals reflecting its current learning status to the generator. Through this iterative interaction, the generator continuously adapts data generation to the evolving requirements of policy optimization. In the following, we detail the generator, selector, and policy, as well as how they interact to form this adaptive training loop.

4.1 The Generator

The generator leverages feedback from both the selector and the policy to identify suitable initial trajectory segments for generation from the original data and generate new data accordingly. Specifically, to select the initial trajectory segment, we maintain a trajectory pool $\mathcal{D}^j = \{\tau_1, \tau_2, \ldots, \tau_N\}$ at the j-th generation round, where each trajectory is associated with feedback signals from the selector and the policy. Each trajectory τ_i consists of multiple sub-trajectory segments $\tau_{t:t+L}$ of length L. For each segment $\tau_{t:t+L} \in \tau_i$, we denote the feedback from the selector and the policy as $\mathbf{s}_t^s$ and $\mathbf{s}_t^p$, respectively (the details of $\mathbf{s}_t^s$ and $\mathbf{s}_t^p$ will be discussed in section 4.2 and 4.3). The generator computes the sampling weight of each segment based on

$$w_t^j = f_{\text{gate}}\left(s_t^s + \lambda_g s_t^p, w_t^{j-1}\right), \tag{1}$$

where λ_g is a hyperparameter that controls the influence of the policy's demand for this type of data on subsequent data generation. A larger λ_g encourages the generator to select the data of type currently needed by the policy, where w_t^j and w_t^{j-1} denote the weights of the sub-trajectory segment $\tau_{t:t+L}$ at the j-th and $(j-1)$-th generation rounds, respectively. f_{gate} denotes a predefined gating function determines how the current evaluation $s_t^s + \lambda_g s_t^p$ adjusts the previous weight w_t^{j-1},

Based on the updated weights, we sample trajectory segments from the trajectory pool $\mathcal{D}^j$ according to their normalized weights:

$$\tau_{t':t'+L} \sim P_{\mathcal{T}}, \qquad P_{\mathcal{T}}(\tau_{t':t'+L}) = \frac{w_{t'}^j}{\sum_{\tau_{k:k+L} \in \mathcal{T}} w_k^j}. \tag{2}$$

where $\tau_{t':t'+L}$ denotes the segments sampled for the j-th generation round. The sampled segment is then used as the initial trajectory context for the generative model to generate subsequent transitions:

$$\tau' = \text{Gen}\left(\tau_{t':t'+L}\right), \tag{3}$$

where $\text{Gen}(\cdot)$ denotes the data generator. PADGen is agnostic to the specific generative model and can, in principle, be instantiated with any data augmentation method that generates subsequent data conditioned on trajectories or transitions. In our implementation, we adopt the generator from TGCVG (Wang et al., 2026), as our focus is on the adaptive generation framework rather than the design of the generative architecture. Given a sampled sub-trajectory segment $\tau_{t':t'+L}$, $\text{Gen}(\cdot)$ generates a continuation conditioned on this segment. The newly generated trajectory τ' is then constructed by concatenating the initial segment $\tau_{t':t'+L}$ with the generated continuation.

Finally, τ' is added back to the trajectory pool: $\mathcal{D}^{j+1} = \mathcal{D}^j \cup \{\tau'\}$,making it available for subsequent data selection and generation.

4.2 The Selector

The selector is designed to evaluate the quality of the data and select samples for policy optimization. The selected data are then used to train the policy, while their quality evaluation scores are retained as additional information and made available to the generator to guide subsequent data generation.

Given a sub-trajectory segment $\tau_{t:t+L} = \{\mathbf{s}_t, \mathbf{a}_t, \ldots, \mathbf{s}_{t+L}, \mathbf{a}_{t+L}\}$ and a set of high-return trajectory segments $\mathcal{D}_e$ ($\mathcal{D}_e \in \mathcal{D}^j$) extracted from the offline dataset, the selector evaluates the quality of the segment according to its state transition pattern and decision pattern. Specifically, the selector computes the transition pattern through the following steps: For a trajectory segment $\tau_{t:t+L}$, we first define its state transition pattern as the sequence of relative state changes with respect to the initial state. Specifically, for each state $\mathbf{s}_{t+i}$, we encode it using a pretrained Hilbert-space representation via $f_h(\mathbf{s}_{t+i})$, where $f_h(\cdot)$ denotes the pretrained Hilbert-space representation encoder (Park et al., 2024). Then, we compute the relative state change as: $\Delta\mathbf{s}_{t+i} = f_h(\mathbf{s}_{t+i}) - f_h(\mathbf{s}_t), i \in$

$\{0, \ldots, L\}$. The resulting sequence $\{\Delta \mathbf{s}_t, \Delta \mathbf{s}_{t+1}, \ldots, \Delta \mathbf{s}_{t+L}\}$ characterizes how the state evolves within the segment, which we refer to as its state transition pattern. The same transformation is applied to every expert segment $\tau^e \in \mathcal{D}_e$, resulting in the corresponding state transition pattern: $\{\Delta \mathbf{s}^e_t, \Delta \mathbf{s}^e_{t+1}, \ldots, \Delta \mathbf{s}^e_{t+L}\}$. The selector then measures the state transition pattern similarity between the given segment and each expert segment:

$$p(\tau) = \frac{1}{|\mathcal{D}_e|L} \sum_{\tau^e \in \mathcal{D}_e} \sum_{k=1}^{L} \mathrm{Sim}\left(\Delta \mathbf{s}_{t+k}, \Delta \mathbf{s}^e_{t+k}\right), \tag{4}$$

where $\mathrm{Sim}(\cdot)$ denotes the similarity function. In this paper, we implement it with cosine similarity. $p(\tau)$ denotes the resulting state transition pattern similarity score of the trajectory segment.

Then, the selector measures how much the decision pattern of the input segment deviates from those of high-return segments based on their state-action pairs:

$$d(\tau) = \frac{1}{|\mathcal{D}_e|L} \sum_{\tau^e \in \mathcal{D}_e} \sum_{k=1}^{L} f_d\left([f_h(\mathbf{s}_{t+k}); \mathbf{a}_{t+k}], [f_h(\mathbf{s}^e_{t+k}); \mathbf{a}^e_{t+k}]\right), \tag{5}$$

where f_d is a predefined deviation metric that measures how much the decision pattern of the input segment deviates from those of high-return segments. A large $f_d(\cdot)$ indicates larger deviations and consequently incurs stronger penalties during selection.

Next, the quality of the sub-trajectory segment $\tau_{t:t+L}$ is therefore computed as:

$$s^s_t = p(\tau) - \lambda_s d(\tau), \tag{6}$$

where λ_s is a hyperparameter. Here, quality refers to compatibility with the high-return reference patterns rather than a direct estimate of ground-truth environment quality. The resulting s^s_t is the selection score provided by the selector, which is used both for candidate selection and as feedback to the generator for subsequent data generation.

Finally, the selector performs quality-aware sampling based on the normalized quality score s^s_t, and the selected sub-trajectory segments are used for policy optimization.

4.3 The Policy

Besides updating the policy using the sub-trajectory segments selected by the selector, the policy additionally provides segment-level optimization feedback to the generator. Specifically, given the trajectory segment $\tau_{t:t+L}$, we associate each trajectory segment with its corresponding policy optimization loss:

$$s^p_t = \mathcal{L}_{\mathrm{policy}}(\tau_{t:t+L}), \tag{7}$$

where s^p_t represents the policy-side feedback signal derived from the optimization loss of the trajectory segment (denoted as policy score). Note that this signal does not directly indicate the quality of a trajectory segment, since a larger loss does not necessarily imply higher value for policy learning. Instead, it reflects the current discrepancy between the policy and the observed trajectory, providing information about which types of data require further adaptation from the policy. The feedback signal is then provided to the generator to guide subsequent data generation. For different policy architectures, $\mathcal{L}_{\mathrm{policy}}(\tau_{t:t+L})$ is implemented according to their corresponding training objectives. For Transformer-based policies and trajectory sequence models (Chen et al., 2021; Janner et al., 2021), we use the masked valid-token action-prediction loss, i.e., the mean squared error (MSE) between the predicted and dataset actions. For Gaussian policies trained through likelihood-based policy regression (Kostrikov et al., 2022; Nair et al., 2020; Peng et al., 2019), we use the negative log-likelihood of the dataset actions under the learned policy distribution. For deterministic policies, following the actor formulation used in TD3+BC (Fujimoto & Gu, 2021), the action-prediction MSE is adopted. For TD3+BC, we use the behavior-cloning residual $(\|\pi_\theta(\mathbf{s}_t) - \mathbf{a}_t\|^2_2)$ as the policy-fitting error. By default, we use actor-only fitting losses without involving critic or value estimates, avoiding the propagation of potential value extrapolation errors into the generation process.

5 Experiments

In this section, we evaluate PADGen from the method, component, and framework levels, and further investigate its limitations. Specifically, we aim to answer the following research questions: (i) How does PADGen compare with existing data augmentation methods(RQ1)? (ii) How do the key components of PADGen contribute to its performance (RQ2)? (iii) How does PADGen compare

with alternative frameworks under low-quality offline data (RQ3)? (iv) What are the limitations of PADGen (RQ4)? We also provide hyperparameter study in Appendix A.1.

5.1 Experimental Setup

Datasets. We evaluate on nine D4RL MuJoCo locomotion tasks: Hopper, HalfCheetah, and Walker2d with Medium, Medium-Replay, and Medium-Expert data. These regimes provide moderate-quality behavior data, broader replay distributions, and mixtures of expert and suboptimal trajectories, respectively, allowing us to test the method under different data quality and diversity.

Baselines. We compare four representative data augmentation approaches: SER (Lu et al., 2023), which uses a diffusion model to generate transitions; TATU (Zhang et al., 2023), which extends the data through short model-based rollouts; DStitch (Li et al., 2024), which constructs complete trajectories by stitching sub-trajectories; and TGCVG (Wang et al., 2026), which generates trajectories by interacting a conservative value-guided policy with a learned dynamics model.

Offline RL Algorithms. Following prior work (Li et al., 2024), we employ three representative offline RL algorithms to evaluate the effectiveness of the data generated by PADGen: DT (Chen et al., 2021), IQL (Kostrikov et al., 2022), and TD3+BC (Fujimoto & Gu, 2021), representing offline RL algorithms based on sequence modeling, implicit policy improvement, and behavior-policy regularization, respectively. This diverse selection allows us to assess the general applicability of PADGen to different offline RL algorithms.

Implementation Details. Since we focus on validating the effectiveness of the overall PADGen framework rather than designing sophisticated module details, we instantiate $f_d(\cdot)$ in Eq. 5 with a simple distance-based formulation, and implement $f_{\text{gate}}(\cdot)$ in Eq. 1 as a predefined linear function with fixed coefficients. This design choice enables a more accurate evaluation of PADGen by avoiding additional complexity introduced by individual module designs. We conduct experiments on NVIDIA A40, each with 48GB of memory. We first train the generator and the Hilbert representation encoder in selector separately, using learning rates of 0.0001 and 0.0003, respectively. Once trained, their parameters are fixed throughout the subsequent process. The generator produces augmented data, from which the selector selects informative samples for training the offline RL algorithms. The resulting policy scores, together with the selection scores, are then fed back to the generator to guide the next round of data generation. For the initial generation round, where feedback from the policy and selector is not yet available, we randomly sample data from the offline dataset as the initial trajectory segments for generation. During the policy-adaptive data generation of PADGen, we set the length of sub-trajectory segment $L = 20$, λ_g in Eq.(1)to 0.5 and λ_s Eq.(6) to 0.5. For the baselines, we generate augmented data following their original settings and combine the generated data with the original offline dataset for policy training.

5.2 Main Results (RQ1)

Table 1 illustrates the experiment results on MuJoCo tasks, and the results report normalized return with the mean and standard deviation over five seeds. From Table 1, we can observe that: (1) PADGen achieves the highest average performance across all three offline RL algorithms, with scores of 82.4, 79.1, and 68.6 on IQL, TD3+BC, and DT, respectively, outperforming their corresponding baselines by 13.4, 16.5, and 10.9 points. Compared with existing augmentation methods, PADGen also achieves the best average performance under every backbone and obtains the best results in 14 out of 18 settings. These results demonstrate that PADGen can be effectively integrated with different offline RL algorithms and consistently improve their overall performance.

(2) Introducing additional augmented data does not necessarily improve policy performance. For example, TATU reduces the average score of IQL from 69.0 to 64.0 and TD3+BC from 62.6 to 58.8, despite providing additional training data. Together with our earlier observation that the utility of training data changes during policy optimization, these results highlight the importance of adapting augmented data to the evolving policy rather than relying on static augmentation.

(3) In several cases where existing augmentation methods provide limited or inconsistent gains, PADGen achieves substantial improvements. For example, on Hopper-M with DT, existing methods yield scores of 60.5–64.4 compared with the baseline score of 62.4, whereas PADGen improves the performance to 85.0. A similar pattern appears on Walker2d-M with IQL, existing augmentation methods achieve scores ranging from 69.6 to 85.9 over a baseline of 81.6, while PADGen further improves the performance to 88.9. This result suggests that policy-adaptive generation can provide useful training data in some settings where conventional augmentation is less effective.

Table 1: Performance on D4RL MuJoCo tasks. Results are normalized returns over five seeds. The rightmost column reports the unweighted average over the six tasks. Cells shaded in darker and lighter gray indicate the best and second-best results, respectively.

Algo	Method	Medium			Med-Replay			Average
		HalfCheetah	Hopper	Walker2d	HalfCheetah	Hopper	Walker2d	
IQL	Baseline	48.3±0.1	65.0±2.9	81.6±0.9	43.9±0.2	96.3±6.5	79.0±3.2	69.0
	SER	49.3±0.1	66.6±2.4	85.9±1.6	46.6±0.1	**102.4**±0.5	85.7±3.6	72.8
	TATU	48.2±0.1	60.3±3.6	76.6±10.7	44.2±0.1	79.6±7.6	75.0±12.1	64.0
	DStitch	49.4±0.1	71.0±4.2	83.2±2.2	44.7±0.1	102.1±0.4	86.6±2.8	72.8
	TGCVG	67.0±0.2	77.3±2.4	69.6±20.1	58.9±0.5	102.2±0.4	95.0±1.6	78.3
	PADGen	**67.2**±0.4	**80.4**±6.9	**88.9**±0.4	**59.5**±0.6	102.3±0.7	**96.0**±1.2	**82.4**
TD3+BC	Baseline	48.4±0.2	59.3±1.1	84.0±0.5	44.5±0.2	59.8±6.7	79.6±4.0	62.6
	SER	48.4±0.4	56.4±1.3	84.9±0.3	45.2±0.1	56.8±9.9	89.1±0.5	63.5
	TATU	48.1±0.2	58.3±4.8	75.8±3.5	44.5±0.3	64.1±10.5	62.1±10.4	58.8
	DStitch	50.4±0.5	60.3±4.9	83.4±1.7	44.7±0.3	79.6±13.5	89.7±4.2	68.0
	TGCVG	66.8±0.4	83.0±1.3	85.7±0.4	58.4±0.5	76.1±3.1	92.1±1.3	77.0
	PADGen	**67.6**±0.2	**84.0**±1.8	**85.9**±0.4	**59.0**±0.2	**85.0**±3.7	**93.5**±0.8	**79.1**
DT	Baseline	42.2±0.1	62.4±3.7	69.8±1.1	34.2±2.2	76.4±4.5	61.2±2.3	57.7
	DStitch	44.2±0.3	60.5±4.3	**72.0**±4.9	**41.0**±0.4	**96.1**±2.0	60.2±1.8	62.3
	TGCVG	42.7±0.1	64.4±1.3	71.9±1.4	39.6±1.2	64.6±4.7	66.5±5.6	58.3
	PADGen	**62.5**±0.2	**85.0**±1.7	69.6±0.8	35.7±10.4	91.6±4.1	**67.2**±3.7	**68.6**

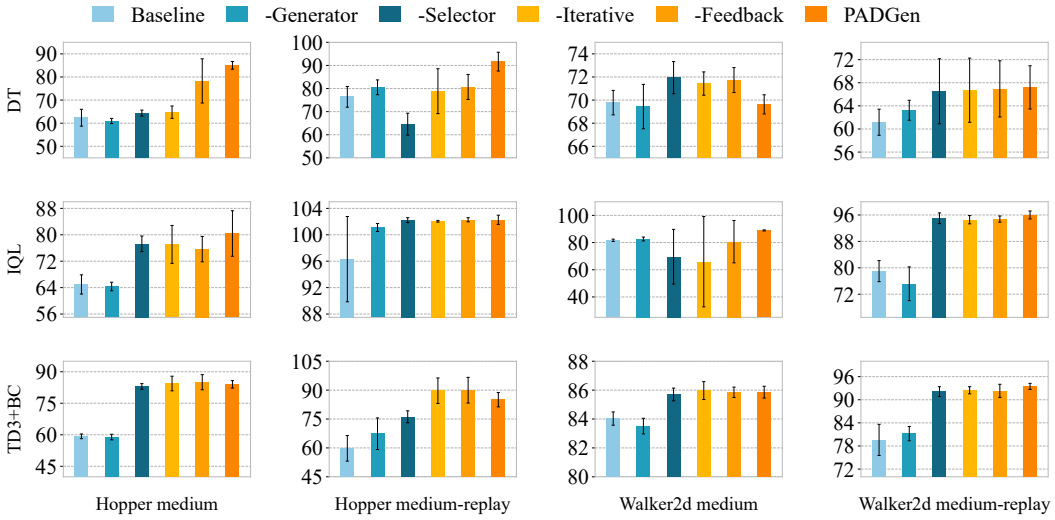


Figure 3: Ablation study on Hopper and Walker2d. The bars represent the mean normalized return over five random seeds, with error bars indicating the standard deviation.

5.3 Ablation Study (RQ2)

To investigate the impact of different components of PADGen, we conduct the ablation study. Specifically, we have the following variants:

- **-Feedback**: Removes the policy and selector feedback to the generator in PADGen.
- **-Iterative**: Based on **-Feedback**, generates the augmented data only once before policy training, instead of adaptively generating data at each iteration.
- **-Selector**: Based on **-Iterative**, further removes the selector, such that all generated data are directly used for policy training without selection.
- **-Generator**: Based on **-Iterative**, further removes the generator and only uses the selector to select data from the original offline dataset for policy training.
- **Baseline**: Further removes the selector from **-Generator**, leaving neither data generation nor data selection; the policy is trained solely on the original offline dataset.

The results are illustrated in Figure 3. From Figure 3, we can observe that: (1) Using only the selector (-Generator) provides limited improvements over the baseline across most tasks. In contrast,

introducing generated data (-Selector) yields substantial gains on several tasks, particularly for IQL and TD3+BC. For example, on Hopper-Medium with TD3+BC, the normalized return increases from 59.26 without augmentation to 83.01 with generated data, while selection alone achieves only 58.91. These results highlight the importance of expanding the offline dataset with generated data, beyond merely selecting samples from the original dataset.

(2) Comparing -Selector with -Iterative, Incorporating the selector provides clear improvements on generator. For example, the normalized return on Hopper-Medium Replay increases from 76.14 to 89.70 with TD3+BC and from 64.58 to 78.85 with DT. These results highlight the importance of distinguishing the utility of generated data before incorporating them into training, demonstrating that the selector complements the generator by identifying more useful samples.

(3) Compared with one-shot generation (-Iterative), iterative generation (-Feedback) yields clear improvements on several tasks; for example, the normalized return on Hopper-Medium increases from 64.77 to 78.30 with DT. This shows that regenerating data throughout policy optimization is generally more effective than generating a fixed augmented dataset before training. The results support our motivation that the data required by the policy can change during optimization and should be updated accordingly.

(4) Incorporating policy and selector feedback further improves performance in several settings; for example, with DT, the normalized return increases from 78.30 to 85.04 on Hopper-Medium and from 80.68 to 91.64 on Hopper-Medium Replay. Although such improvements are not consistent across all tasks, the results demonstrate that feedback can provide useful guidance for adapting data generation during policy optimization.

5.4 Robust to low-quality Data (RQ3)

To evaluate the robustness of PADGen framework to low-quality offline data, we progressively degrade Hopper-Medium-Replay by mixing it with random data at ratios of 25%, 50%, and 75%. Unlike the main experiments that compare PADGen with existing augmentation methods, this controlled setting focuses on investigating how PADGen differs from alternative data processing frameworks in handling low-quality data. Therefore, we consider two representative strategies for handling low-quality data: selection-only, which filters the available offline data, and augmentation-only, which expands the dataset without adaptive selection.

Table 2: Results on Hopper-medium-replay mixtures. The columns denote the proportion of random data in the training data. SelOnly and GenOnly denote the selection-only and generation-only variants, respectively. Cells shaded in darker and lighter gray indicate the best and second-best results, respectively.

Algo	Method	25% Rand	50% Rand	75% Rand
DT	Original	67.93 ± 8.16	45.38 ± 10.65	53.47 ± 4.51
	SelOnly	78.80 ± 5.03	60.87 ± 6.10	57.28 ± 6.21
	GenOnly	63.02 ± 11.31	50.01 ± 12.16	46.37 ± 8.84
	PADGen	**82.00±1.90**	**70.01±2.99**	**62.75±2.94**
IQL	Original	75.9 ± 10.7	69.5 ± 6.3	71.3 ± 14.3
	SelOnly	78.7 ± 13.5	71.1 ± 7.2	69.7 ± 9.9
	GenOnly	18.5 ± 7.7	6.8 ± 5.0	14.0 ± 15.8
	PADGen	**91.2±7.7**	**90.8±2.7**	**90.4±2.4**

The results shown in Table 2, from which we can observe that:

(1) Simply augmenting the available dataset does not reliably address the presence of low-quality data. For DT, augmentation-only obtains 63.02, 50.01, and 46.37 under the 25%, 50%, and 75% random-data mixtures, respectively. For IQL, its performance decreases further to 18.5, 6.8, and 14.0. In several cases, augmentation-only performs worse than the original data, suggesting that indiscriminately adding generated samples can amplify the limitations of the data on which generation is based.

(2) Selection provides a more effective way of utilizing degraded offline data than augmentation alone. Selection-only consistently outperforms augmentation-only for both learners and all three mixture ratios. For DT, it achieves 78.80, 60.87, and 57.28, compared with 63.02, 50.01, and 46.37 for augmentation-only. For IQL, the corresponding results are 78.7, 71.1, and 69.7, compared with 18.5, 6.8, and 14.0. Selection-only also improves over the original data in several settings, although this improvement is not universal. This indicates that selection can better identify useful samples, but remains constrained by the information and coverage already present in the offline dataset.

(3) Selection alone is inherently limited by the fixed support of the offline data. PADGen addresses this limitation by coupling data selection with adaptive data generation. It achieves the best performance for both learners across all three mixture ratios. Compared with Selection-only, PADGen improves the DT results by 3.20, 9.14, and 5.47 points, and the IQL results by 12.5, 19.7, and 20.7 points under the 25%, 50%, and 75% random-data mixtures, respectively. These results show that the benefit of PADGen cannot be attributed merely to generating additional data or selecting existing data. Instead, its advantage arises from integrating the two processes: selection helps identify data that are more useful for policy learning, while adaptive generation expands the training distribution beyond the samples already available in the offline dataset. This combination is particularly beneficial when the original dataset contains a substantial amount of low-quality data.

5.5 LIMITATIONS (RQ4)

While PADGen generally improves policy learning when the offline dataset contains suboptimal or low-quality data, its benefits become less consistent on expert datasets. As shown in Table 3, PADGen improves the average performance of IQL from 97.4 to 101.0, but decreases that of DT and TD3+BC from 101.8 to 98.8 and from 98.9 to 83.1, respectively. The degradation is particularly pronounced on HalfCheetah-Expert with TD3+BC.

Table 3: Performance on D4RL MuJoCo medium-expert tasks. Results are normalized returns over five seeds. The rightmost column reports the unweighted average over the three tasks. Cells shaded in darker and lighter gray indicate the best and second-best results, respectively.

Algo	Method	Medium-Expert			Average
		HalfCheetah	Hopper	Walker2d	
DT	Baseline	**89.8±1.0**	**107.8±1.2**	**107.7±0.2**	**101.8**
	TGCVG	87.7±4.1	106.1±2.9	106.7±1.2	100.2
	PADGen	84.0±2.8	104.8±1.3	107.5±0.3	98.8
IQL	Baseline	**92.9±0.2**	87.6±47.5	**111.5±1.1**	97.4
	TGCVG	86.3±3.2	100.5±14.5	111.1±0.2	99.3
	PADGen	86.6±3.4	**105.4±5.3**	110.9±0.2	**101.0**
TD3+BC	Baseline	**89.9±1.7**	96.6±3.3	110.2±0.2	**98.9**
	TGCVG	81.4±9.7	92.7±5.0	**110.3±0.2**	94.8
	PADGen	42.1±23.7	**97.0±5.0**	**110.3±0.2**	83.1

We conjecture that this limitation stems from both generation and selection. First, expert datasets already contain highly informative trajectories, leaving limited room for the generator to produce data with higher utility. Consequently, generated samples may provide little additional information and can even be less useful than the original expert data. Second, although the selector can mitigate this issue by identifying useful generated samples, selection itself becomes more challenging in the expert regime. When the original data are already of high quality, the differences in utility among candidate samples may become less pronounced, making it more difficult to identify generated samples that provide additional benefits over the existing data. These observations suggest that PADGen is most beneficial when there remains a meaningful data-utility gap that can be improved through adaptive generation and selection. As the original dataset approaches expert-level quality, this gap becomes smaller, thereby limiting the potential gains of PADGen.

6 CONCLUSION

In this work, we revisit data augmentation for offline RL from the perspective of evolving data utility. We show that the utility of training data can change as policy optimization proceeds, revealing a fundamental limitation of the conventional static generation paradigm, where augmented data remain fixed throughout training. To address this issue, we propose PADGen, an adaptive data generation framework that couples data generation, data selection, and policy optimization in a closed loop. By incorporating feedback from both the selector and the evolving policy, PADGen continuously adapts the generated data to the current needs of policy learning. Extensive experiments across different offline RL algorithms and datasets demonstrate that PADGen consistently improves policy performance over existing data augmentation methods. Further analyses show that its generation and selection mechanisms play complementary roles, and their integration is particularly effective when the underlying offline data contain substantial low-quality samples. Meanwhile, our results on expert datasets reveal that the benefits of PADGen become less consistent when the original data are already highly informative, suggesting that PADGen is most beneficial when there remains a meaningful data-utility gap that can be improved through adaptive generation and selection. These findings highlight the importance of considering the evolving utility of data in offline RL and provide a promising direction toward policy-adaptive data augmentation.

REFERENCES

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

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, volume 34, pp. 15084–15097, 2021.

Nicholas E. Corrado, Yuxiao Qu, John U. Balis, Adam Labiosa, and Josiah P. Hanna. Guided data augmentation for offline reinforcement learning and imitation learning. *Reinforcement Learning Journal*, 1:198–215, 2024.

Mehdi Fatemi, Mary Wu, Jeremy Petch, Walter Nelson, Stuart J. Connolly, Alexander Benz, Anthony Carnicelli, and Marzyeh Ghassemi. Semi-markov offline reinforcement learning for healthcare. In *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pp. 119–137. PMLR, 2022.

Mattie Fellows, Clarisse Wibault, Uljad Berdica, Johannes Forkel, Michael A. Osborne, and Jakob Nicolaus Foerster. Fully offline reinforcement learning. *Reinforcement Learning Journal*, 7, 2026.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pp. 2052–2062. PMLR, 2019.

Xingshuai Huang, Di Wu, and Benoit Boulet. Goal-conditioned data augmentation for offline reinforcement learning. *arXiv preprint arXiv:2412.20519*, 2024.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, volume 34, pp. 1273–1286, 2021.

Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 9902–9915. PMLR, 2022.

Min Jiang, Edward Grefenstette, and Tim Rocktaschel. Prioritized level replay. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pp. 4940–4950. PMLR, 2021.

Chinmaya Kausik, Yangyi Lu, Kevin Tan, Maggie Makar, Yixin Wang, and Ambuj Tewari. Offline policy evaluation and optimization under confounding. In *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238, pp. 1459–1467. PMLR, 2024.

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. MOReL: Model-based offline reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

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

Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

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, 2020.

Aviral Kumar, Anikait Singh, Stephen Tian, Chelsea Finn, and Sergey Levine. A workflow for offline model-free robotic reinforcement learning. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 417–428. PMLR, 2022.

Dongsu Lee, Chanin Eom, and Minhae Kwon. AD4RL: Autonomous driving benchmarks for offline reinforcement learning with value-based dataset. In *2024 IEEE International Conference on Robotics and Automation*, 2024a. doi: 10.1109/ICRA57147.2024.10610308.

Jaewoo Lee, Sujin Yun, Taeyoung Yun, and Jinkyoo Park. Gta: Generative trajectory augmentation with guidance for offline reinforcement learning. *Advances in neural information processing systems*, 37:56766–56801, 2024b.

Kyowoon Lee and Jaesik Choi. State-covering trajectory stitching for diffusion planners. *Advances in Neural Information Processing Systems*, 38:57273–57303, 2026.

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.

Guanghe Li, Yixiang Shan, Zhengbang Zhu, Ting Long, and Weinan Zhang. Diffstitch: Boosting offline reinforcement learning with diffusion-based trajectory stitching. *arXiv preprint arXiv:2402.02439*, 2024.

Cong Lu, Philip Ball, Yee Whye Teh, and Jack Parker-Holder. Synthetic experience replay. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46323–46344, 2023.

Jiafei Lyu, Xiu Li, and Zongqing Lu. Double check your state before trusting it: Confidence-aware bidirectional offline model-based imagination. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

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

Soichiro Nishimori, Xin-Qiang Cai, Johannes Ackermann, and Masashi Sugiyama. Offline reinforcement learning with domain-unlabeled data. *Reinforcement Learning Journal*, 6:1773–1793, 2025.

Seohong Park, Tobias Kreiman, and Sergey Levine. Foundation policies with hilbert representations. *arXiv preprint arXiv:2402.15567*, 2024.

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

Yunpeng Qing, Yixiao Chi, Shuo Chen, Shunyu Liu, Kexuan Zhou, Sixu Lin, Litao Liu, and Changqing Zou. Bitrajdiff: Bidirectional trajectory generation with diffusion models for offline reinforcement learning. *arXiv preprint arXiv:2506.05762*, 2025.

Rafael Rafailov, Kyle Hatch, Anikait Singh, Laura Smith, Aviral Kumar, Ilya Kostrikov, Philippe Hansen-Estruch, Victor Kolev, Philip Ball, Jiajun Wu, Chelsea Finn, and Sergey Levine. D5RL: Diverse datasets for data-driven deep reinforcement learning. *arXiv preprint arXiv:2408.08441*, 2024.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016.

Tobias Schmähling, Matthias Burkhardt, and Tobias Windisch. Trajectory-level data augmentation for offline reinforcement learning. *arXiv preprint arXiv:2605.13401*, 2026.

Kajetan Schweighofer, Marius-Constantin Dinu, Andreas Radler, Markus Hofmarcher, Vihang Prakash Patil, Angela Bitto-Nemling, Hamid Eghbal-Zadeh, and Sepp Hochreiter. A dataset perspective on offline reinforcement learning. In *Proceedings of the 1st Conference on Lifelong Learning Agents*, volume 199, pp. 470–517. PMLR, 2022.

Tianyu Shi, Dong Chen, Kaian Chen, and Zhaojian Li. Offline reinforcement learning for autonomous driving with safety and exploration enhancement. *arXiv preprint arXiv:2110.07067*, 2021.

Samarth Sinha, Ajay Mandlekar, and Animesh Garg. S4rl: Surprisingly simple self-supervision for offline reinforcement learning in robotics. In *Conference on Robot Learning*, pp. 907–917. PMLR, 2022.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018.

Jianhao Wang, Wenzhe Li, Haozhe Jiang, Guangxiang Zhu, Siyuan Li, and Chongjie Zhang. Offline reinforcement learning with reverse model-based imagination. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

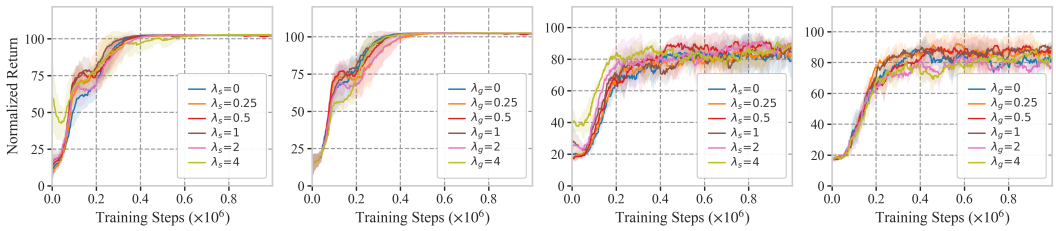
Tieru Wang, Kunbao Wu, and Guoshun Nan. Trajectory generation with conservative value guidance for offline reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 28623–28641, 2026.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Junjie Zhang, Jiafei Lyu, Xiaoteng Ma, Jiangpeng Yan, Jun Yang, Le Wan, and Xiu Li. Uncertainty-driven trajectory truncation for data augmentation in offline reinforcement learning. *arXiv preprint arXiv:2304.04660*, 2023. doi: 10.3233/FAIA230618.

Doudou Zhou, Yufeng Zhang, Aaron Sonabend-W, Zhaoran Wang, Junwei Lu, and Tianxi Cai. Federated offline reinforcement learning. *Journal of the American Statistical Association*, 119 (548):3152–3163, 2024. doi: 10.1080/01621459.2024.2310287.



(a) IQL: λ_s (b) IQL: λ_g (c) TD3+BC: λ_s (d) TD3+BC: λ_g

Figure 4: Hyperparameter study.

A Appendix

A.1 Hyperparameter Analysis

We investigate the sensitivity of PADGen to two key hyperparameters, λ_s and λ_g, using TD3+BC and IQL as the underlying offline RL learners. Specifically, λ_s controls the penalty on decision-pattern deviations during data selection, where a larger value imposes a stronger penalty on segments whose decision patterns deviate from those of high-return segments. In contrast, λ_g controls the weight of policy-loss feedback during data generation, where a larger value makes the generator place greater emphasis on the learning signal from the current offline RL policy. As shown in Figure 4, PADGen maintains relatively stable performance across a broad range of both hyperparameters. The sensitivity varies across offline RL learners: IQL remains highly stable under different weight values, whereas TD3+BC exhibits relatively larger variations. Notably, setting either λ_s or λ_g to zero results in lower performance on TD3+BC, indicating that both decision-pattern guidance for selection and policy-dependent feedback for generation can contribute to effective data adaptation. These results suggest that PADGen is generally robust to the choices of λ_s and λ_g, although their influence can vary with the underlying offline RL learner.