A. Appendix

B. More Experiments

B.1. Single Behavior Policy v.s. Multiple Behavior Policies

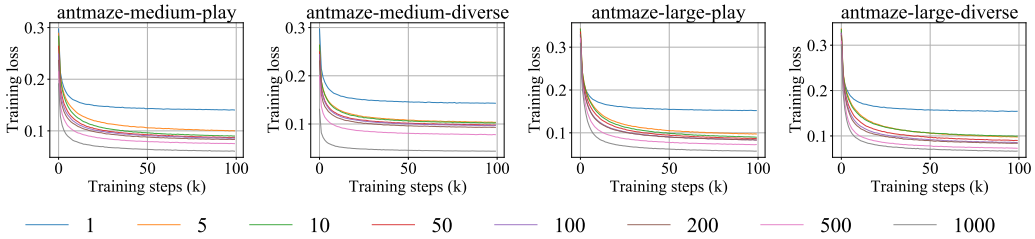

Figure 1. Learning curves of behavior cloning on AntMaze suites (*-v2) in D4RL, where the x-axis denotes the training steps, and the y-axis denotes the training loss. The number N in the legend denotes the number of sub-tasks. If $N = 1$, we learn a single behavior policy for the whole offline dataset.

In Figure 1, we provide empirical evidence that learning a single behavior policy (using BC) is not sufficient to characterize the whole offline dataset, and multiple behavior policies (conducting task decomposition) deliver better resilience to characterize the offline data than a single behavior policy.

B.2. Decomposition Rules

In DROP algorithm, we explicitly decompose an offline task into multiple sub-tasks, over which we then reframe the offline policy learning problem as one of offline model-based optimization. In this section, we discuss three different designs for the task decomposition rule.

Random(N, M)**:** We decomposition offline dataset $\mathcal{D} := \{\tau\}$ into N subsets, each of which contains at most M trajectories that are randomly sampled from the offline dataset.

Quantization(N, M)**:** Leveraging the returns of trajectories in offline data, we first quantize offline trajectories into N bins, and then randomly sample at most M trajectories (as a sub-task) from each bin. Specifically, in the i-th bin, the quantized trajectories $\{\tau_i\}$ satisfy $\mathrm{R}_{\min} + \Delta * i < \mathrm{Return}(\tau_i) \leq \mathrm{R}_{\min} + \Delta * (i+1)$, where $\Delta = \frac{(\mathrm{R}_{\max} - \mathrm{R}_{\min})}{N}$, $\mathrm{Return}(\tau_i)$ denotes the return of trajectory τ_i, and $\mathrm{R}_{\max}$ and $\mathrm{R}_{\min}$ denote the maximum and minimum trajectory returns in the offline dataset respectively.

Rank(N, M)**:** We first rank the offline trajectories descendingly based on their returns, and then sequentially sample M trajectories for each subset. (*We adopt this decomposition rule in main paper.*)

In Figure 2, we provide the comparison of the above three decomposition rules (see the selected number of sub-tasks and the number of trajectories in each sub-task in Table 5). We can find that across a variety of tasks, the decomposition rule has a fundamental impact on the subsequent model-based optimization. Across different tasks and different embedding inference rules, Random and Quantization decomposition rules tend to exhibit large performance fluctuations, which reveals the importance of choosing a suitable task decomposition rule. In our paper, we adopt the Rank decomposition rule, as it demonstrates a more robust performance shown in Figure 2. In Appendix B.4, we adopt the conditional variational auto-encoder (CVAE) to conduct automatic task decomposition (treating each trajectory in offline dataset as an individual task) and we find such implementation (DROP+CVAE) can further improve DROP's performance. In future work, we also encourage better decomposition rules to decompose offline tasks so as to enable more effective model-based optimization for offline RL tasks.

Comparison with filtered behavior cloning. We also note that the Rank decomposition rule leverages more high-quality trajectories than the other two decomposition rules (Random and Quantization). Thus, a natural question to ask is, is the performance of Rank better than that of Random and Quantization due to the presence of more high-quality trajectories in the decomposed sub-tasks? That is, whether DROP (using the Rank decomposition rule) only conducts behavioral cloning over those high-quality trajectories, thus leading to better performance.

To answer the above question, we compare DROP (using the Rank decomposition rule) with filtered behavior cloning

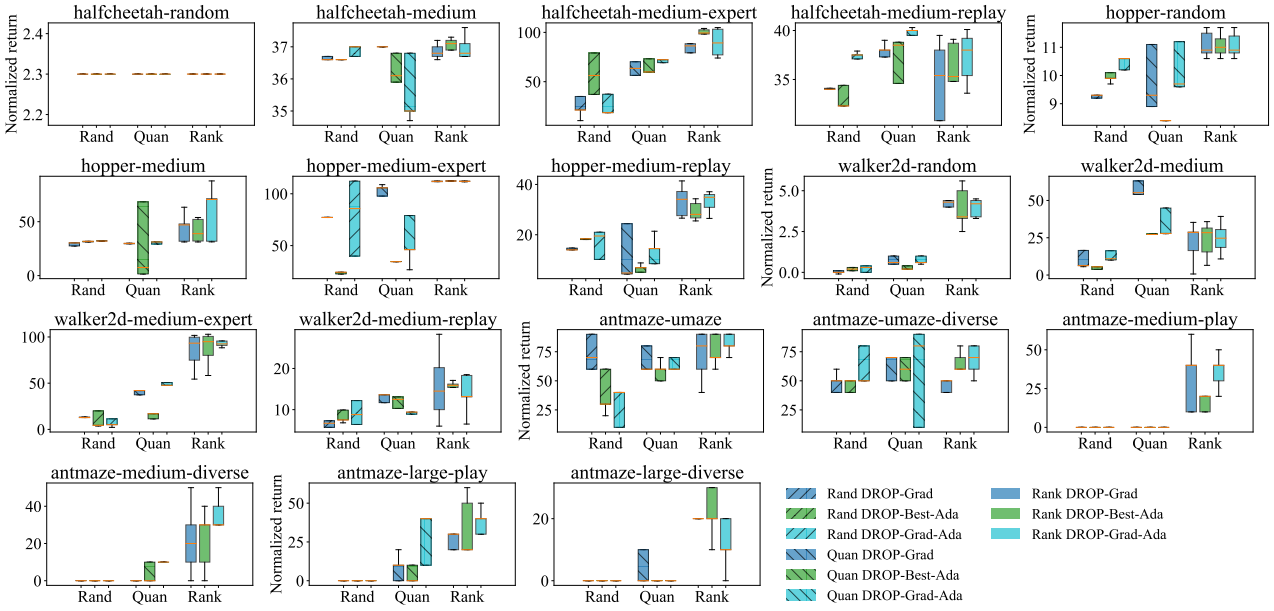

Figure 2. Comparison of three different decomposition rules on D4RL MuJoCo-Gym suite (*-v0) and AntMaze suite (*-v2), where "Rand", "Quan" and "Rank" denote the Random, Quantization, and Rank decomposition rules respectively. We can find across 18 tasks (AntMaze and MuJoCo-Gym suites) and 3 embedding inference methods (DROP-Grad, DROP-Best-Ada, and DROP-Grad-Ada), Rank is more stable and yields better performance compared with the other two decomposition rules.

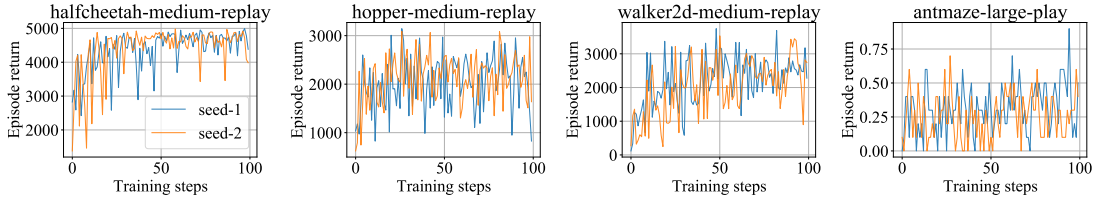

Figure 3. Learning curves of DROP, where the x-axis denotes the training steps (k), the y-axis denotes the evaluation return (using DROP-Best embedding inference rule). We only show two seeds for legibility.

(F-BC), where the latter (F-BC) performs behavior cloning after filtering for trajectories with high returns. We provide the comparison results in Table 1. We can find that in AntMaze tasks, the overall performance of DROP is higher than that of F-BC. For the MuJoCo-Gym suite, DROP-based methods outperform F-BC on these offline tasks that contain plenty of sub-optimal trajectories, including the random, medium, and medium-replay domains. This result indicates that DROP can leverage embedding inference (extrapolation) to find a better policy beyond all the behavior policies in sub-tasks, which is more effective than simply performing imitation learning on a subset of the dataset.

B.3. Online Fine-tuning

Online fine-tuning (checkpoint-level). In Figure 3, we show the learning curves of DROP-Best on four DR4L tasks. We can find that DROP exhibits a high variance (in performance) across training steps[1], which means the performance of the agent may be dependent on the specific stopping point chosen for evaluation (such instability also exists in prior offline RL methods, Fujimoto and Gu (2021)).

To choose a suitable stopping checkpoint over which we perform the DROP inference (DROP-Grad, DROP-Best-Ada and DROP-Grad-Ada), we propose to conduct *checkpoint-level* online fine-tuning (see Algorithm 1 in Section C for more details): we evaluate each of the latest T checkpoint models and choose the best one that leads to the highest episode return.

[1]In view of such instability, we evaluate our methods over multiple checkpoints for each seed, instead of choosing the final checkpoint models during the training loop (see the detailed evaluation protocol in Appendix C).

Table 1. Comparison between our DROP (using the Rank decomposition rule) and filtered behavior cloning (F-BC) on D4RL AntMaze and MuJoCo suites (*-v2). We take the baseline results of BC and F-BC from Emmons et al. (2021), where F-BC is trained over the top 10% trajectories, ordered by the returns. Our DROP results are computed over 5 seeds and 10 episodes for each seed.

Tasks		BC	F-BC	DROP-Grad	DROP-Best-Ada	DROP-Grad-Ada
antmaze-umaze		54.6	60	72 ± 17.2	78 ± 11.7	80 ± 12.6
antmaze-umaze-diverse		45.6	46.5	48 ± 22.3	62 ± 16	66 ± 12
antmaze-medium-play		0	42.1	24 ± 10.2	34 ± 12	30 ± 21
antmaze-medium-diverse		0	37.2	20 ± 19	24 ± 12	30 ± 16.7
antmaze-large-play-		0	28	24 ± 8	36 ± 17.4	42 ± 17.2
antmaze-large-diverse		0	34.3	14 ± 8	20 ± 14.1	26 ± 13.6
halfcheetah	random	2.3	2	2.3 ± 0	2.3 ± 0	2.3 ± 0
hopper	random	4.8	4.1	5.1 ± 0.8	5.4 ± 0.7	5.5 ± 0.6
walker2d	random	1.7	1.7	2.8 ± 1.7	3 ± 1.6	3 ± 1.8
halfcheetah	medium	42.6	42.5	42.4 ± 0.7	42.9 ± 0.4	43.1 ± 0.4
hopper	medium	52.9	56.9	57.5 ± 6.4	60.3 ± 6.1	59.5 ± 5.1
walker2d	medium	75.3	75	76.5 ± 2.4	75.8 ± 3	79.1 ± 1.4
halfcheetah	medium-replay	36.6	40.6	39.5 ± 1	40.4 ± 0.8	40.3 ± 1.2
hopper	medium-replay	18.1	75.9	48 ± 17.7	83.4 ± 6.5	87.4 ± 2.1
walker2d	medium-replay	26	62.5	37.4 ± 13.5	60.9 ± 7.4	61.9 ± 2.3

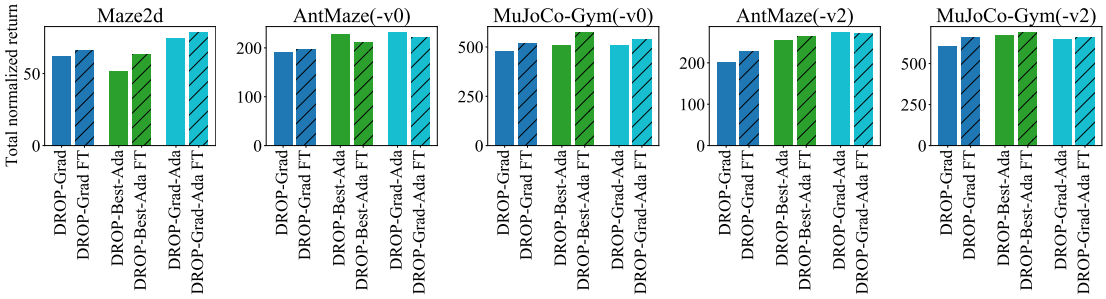

Figure 4. Total normalized returns across all the tasks in Maze2d, AntMaze, and MuJoCo-Gym suites.

In Figure 4, we show the total normalized returns across all the tasks in each suite (including Maze2d, AntMaze, and MuJoCo-Gym). We can find that in most tasks, fine-tuning (FT) can guarantee performance improvement. However, we also find such fine-tuning causes negative impacts on performance in AntMaze(*-v0) suite. The main reason is that, in this checkpoint-level fine-tuning, we choose the "suitable" checkpoint model using the DROP-Best embedding inference rule, while we adopt the other three embedding inference rules (DROP-Grad, DROP-Best-Ada and DROP-Grad-Ada) at the test time. Such a finding also implies that the success of DROP's test-time adaptation is not entirely dependent on the best embedding across sub-tasks [2] (*i.e.*, the best embedding $\mathbf{z}_0^*(\mathbf{s}_0)$ in DROP-Best), but requires switching between some "suboptimal" embeddings (using DROP-Best-Ada) or extrapolating new embeddings (using DROP-Grad-Ada).

Online fine-tuning (embedding-level). Beyond the above checkpoint-level fine-tuning procedure, we can also conduct *embedding-level* online fine-tuning: we aim to choose a suitable gradient update step for the gradient-based embedding inference rules (including DROP-Grad and DROP-Grad-Ada). Similar to the checkpoint-level fine-tuning, we first conduct the test-time adaptation procedure (DROP-Grad and DROP-Grad-Ada) over a set of gradient update steps, and then choose the best step that leads to the highest episode return (see Algorithm 2 in Section C for more details).

[2]Conversely, if the performance of DROP depends on the best embedding across sub-tasks (*i.e.*, $\mathbf{z}_0^*(\mathbf{s}_0)$ in DROP-Best), then the checkpoint model we choose by fine-tuning with DROP-Best should enable a consistent performance improvement for rules that perform embedding inference with DROP-Best-Ada and DROP-Grad-Ada. However, we find a performance drop in AntMaze(*-v0) suite, which means there is no explicit dependency between the best embedding $\mathbf{z}_0^*(\mathbf{s}_0)$ and the inferred embedding using the adaptive inference rules.

Table 2. Online fine-tuning results (initial performance → performance after online fine-tuning). The baseline results of AWAC, CQL, and IQL are taken from Kostrikov et al. (2021), where they run 1M online steps to fine-tune the learned policy. For our DROP method (DROP-Grad and DROP-Grad-Ada), we run 0.3M ($= 6_{\text{checkpoint}} \times 50_{K_{\max}} \times 1000_{\text{steps per episode}}$) online steps to fine-tune (embedding-level) the policy, *i.e.*, aiming to find the optimal gradient ascent step that is used to infer the contextual embedding $\mathbf{z}^*(\mathbf{s}_0)$ or $\mathbf{z}^*(\mathbf{s}_t)$ for $\pi^*(\mathbf{a}_t|\mathbf{s}_t) := \beta(\mathbf{a}_t|\mathbf{s}_t, \cdot)$ (see Algorithm 2 for the details). Moreover, for medium-* and large-* tasks, we conduct additional parametric-level fine-tuning, with 0.7M online steps to update the policy's parameters.

Task (*-v0)	AWAC	CQL	IQL	DROP-Grad	DROP-Grad-Ada
umaze	56.7 → 59	70.1 → 99.4	86.7 → 96	70 → 96 ± 1.2	76 → 98 ± 0
umaze-diverse	49.3 → 49	31.1 → 99.4	75 → 84	54 → 88 ± 8	66 → 94 ± 4.9
medium-play	0 → 0	23 → 0	72 → 95	20 → 56 ± 8.9	30 → 50 ± 6.3 → 94 ± 2.9
medium-diverse	0.7 → 0.3	23 → 32.3	68.3 → 92	12 → 44 ± 4.9	22 → 38 ± 4.9 → 96 ± 0.8
large-play	0 → 0	1 → 0	25.5 → 46	16 → 38 ± 8.9	16 → 40 ± 6.3 → 53 ± 1.3
large-diverse	1 → 0	1 → 0	42.6 → 60.7	20 → 40 ± 13.6	22 → 46 ± 10.2 → 58 ± 4.5
	$\underbrace{\rightarrow}_{1M}$	$\underbrace{\rightarrow}_{1M}$	$\underbrace{\rightarrow}_{1M}$	$\underbrace{\rightarrow}_{0.3M}$	$\underbrace{\rightarrow}_{0.3M}$ $\underbrace{\rightarrow}_{0.7M}$

In Table 2, we compare our DROP (DROP-Grad and DROP-Grad-Ada) to three offline RL methods (AWAC (Nair et al., 2020), CQL (Kumar et al., 2020) and IQL (Kostrikov et al., 2021)), reporting the initial performance and the performance after online fine-tuning. We can find that the embedding-level fine-tuning (0.3M) enables a significant improvement in performance. The fine-tuned DROP-Grad-Ada (0.3M) outperforms the AWAC and CQL counterparts in most tasks, even though we take fewer rollout steps to conduct the online fine-tuning (baselines take 1M online rollout steps, while DROP-based fine-tuning takes 0.3M steps). However, there is still a big gap between the fine-tuned IQL and the embedding-level fine-tuned DROP (0.3M). Considering that there remain 0.7M online steps in the comparison, we further conduct "parametric-level" fine-tuning (updating the parameters of the policy network) for our DROP-Grad-Ada on medium-* and large-* tasks, we can find which achieves competitive fine-tuning performance even compared with IQL.

B.4. DROP + CVAE Implementation

CVAE-based embedding learning. Similar to LAPO (Chen et al., 2022) and PLAS (Zhou et al., 2020), we adopt the conditional variational auto-encoder (CVAE) to model offline data. Specifically, we learn the contextual policy and behavior embedding:

$$\beta(\mathbf{a}|\mathbf{s},\mathbf{z}), \phi(\mathbf{z}|\mathbf{s}) \leftarrow \arg\max_{\beta,\phi} \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\mathcal{D}} \mathbb{E}_{(\mathbf{z})\sim\phi(\mathbf{z}|\mathbf{s})} \big[\log \beta(\mathbf{a}|\mathbf{s},\mathbf{z})\big] - \mathrm{KL}(\phi(\mathbf{z}|\mathbf{s}) \| p(\mathbf{z})). \tag{1}$$

Then, we learn the score model f with the TD-error and the conservative regularization:

$$f \leftarrow \arg\min_{f} \mathbb{E}_{(\mathbf{s},\mathbf{a},\mathbf{s}',\mathbf{a}')\sim\mathcal{D}} \left[\big(R(\mathbf{s},\mathbf{a}) + \gamma \bar{f}(\mathbf{s}', \mathbf{a}', \phi(\mathbf{z}|\mathbf{s})) - f(\mathbf{s},\mathbf{a},\phi(\mathbf{z}|\mathbf{s})) \big)^2 \right], \tag{2}$$
$$\text{s.t. } \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{z}\sim\mu(\mathbf{z}),\mathbf{a}\sim\beta(\mathbf{a}|\mathbf{s},\mathbf{z})} [f(\mathbf{s},\mathbf{a},\mathbf{z})] - \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{z}\sim\phi(\mathbf{z}|\mathbf{s}),\mathbf{a}\sim\beta(\mathbf{a}|\mathbf{s},\mathbf{z})} [f(\mathbf{s},\mathbf{a},\mathbf{z})] \le \eta,$$

where $\bar{f}$ denotes a target network and $\mu(\mathbf{z})$ denotes the uniform distribution over the $\mathcal{Z}$-space.

In testing, we also dynamically adapt the outer-level optimization, setting policy inference with $\pi^*(\mathbf{a}|\mathbf{s}) = \beta(\mathbf{a}|\mathbf{s}, \mathbf{z}^*(\mathbf{s}))$, where $z^*(\mathbf{s}) = \arg\max_z f\big(\mathbf{s}, \beta(\mathbf{a}|\mathbf{s},\mathbf{z}), \mathbf{z}\big)$.

B.5. Ablation Study

Note that our embedding inference depends on the learned score model f. Without proper regularization, such inference will lead to out-of-distribution embeddings that are erroneously high-scored (Q2). Here we conduct an ablation study to examine the impact of the conservative regularization used for learning the score model.

In Figure 5, we compare DROP-Grad and DROP-Grad-Ada to their naive implementation (*w/o Reg*) that ablates

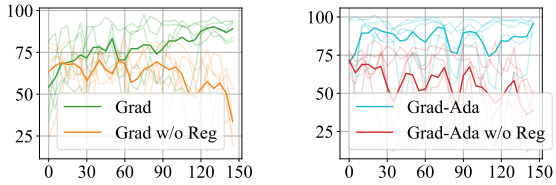

Figure 5. Ablation on the conservative regularization. The y-axis represents the normalize return, and the x-axis represents the number of gradient-ascent steps used for embedding inference at deployment. We plot each random seed as a transparent line; the solid line corresponds to the average across 5 seeds.

Table 3. Hyper-parameters of the task embedding function $\phi(\mathbf{z}|\mathbf{s})$, the contextual behavior policy $\beta(\mathbf{a}|\mathbf{s},\mathbf{z})$, and the score function $f(\mathbf{s},\mathbf{a},\mathbf{z})$. The task embedding function $\phi(\mathbf{z}|\mathbf{s})$: $\mathbf{z} \leftarrow \text{Enc\_0}(n)$. The contextual behavior policy $\beta(\mathbf{a}|\mathbf{s},\mathbf{z})$: $\mathbf{a} \leftarrow \text{Enc\_2}(\mathbf{z}, \text{Enc\_1}(\mathbf{s}))$. The score function $f(\mathbf{s},\mathbf{a},\mathbf{z})$: $f \leftarrow \text{Enc\_4}(\mathbf{z}, \text{Enc\_3}(\mathbf{s},\mathbf{a}))$.

	Enc_0	Enc_1	Enc_2	Enc_3	Enc_4
Optimizer	Adam	Adam	Adam	Adam	Adam
Hidden layer	2	2	3	2	3
Hidden dim	512	512	512	512	512
Activation function	ReLU	ReLU	ReLU	ReLU	ReLU
Learning rate	1.00E-03	1.00E-03	1.00E-03	1.00E-03	1.00E-03
Mini-batch size	1024	1024	1024	1024	1024

the regularization on halfcheetah-medium-expert. We can find that removing the conservative regularization leads to unstable performance when changing the update steps of gradient-based optimization. However, we empirically find that in some tasks such a naive implementation (*w/o Reg*) does not necessarily bring unstable inference (Appendix D). Although improper gradient update step leads to faraway embeddings, to some extent, embedding-conditioned behavior policy can correct such deviation.

C. Implementation Details

For the practical implementation of DROP, we parameterize the task embedding function $\phi(\mathbf{z}|n)$, the contextual behavior policy $\beta(\mathbf{a}|\mathbf{s},\mathbf{z})$, and the score model $f(\mathbf{s},\mathbf{a},\mathbf{z})$ with neural networks (see Appendix C for specific architectures). For Equation 8 in the main paper, we construct a Lagrangian and solve the optimization through primal-dual gradient descent. For the choice of $\mu(\mathbf{z})$, we simply set $\mu(\mathbf{z})$ to be the uniform distribution over the $\mathcal{Z}$-space and empirically find that such uniform sampling can effectively avoid the out-of-distribution extrapolation at inference.

Lagrangian relaxation. To optimize the constrained objective in Equation 8 in the main paper, we construct a Lagrangian and solve the optimization through primal-dual gradient descent,

$$\min_{f}\max_{\lambda>0} \quad \mathbb{E}_{\mathcal{D}_n\sim\mathcal{D}_{[N]}}\mathbb{E}_{(\mathbf{s},\mathbf{a},\mathbf{s}',\mathbf{a}')\sim\mathcal{D}_n}\left[\left(R(\mathbf{s},\mathbf{a})+\gamma\bar{f}(\mathbf{s}',\mathbf{a}',\phi(\mathbf{z}|n))-f(\mathbf{s},\mathbf{a},\phi(\mathbf{z}|n))\right)^2\right]+$$
$$\lambda\left(\mathbb{E}_{n,\mu(\mathbf{z})}\mathbb{E}_{\mathbf{s}\sim\mathcal{D}_n,\mathbf{a}\sim\beta(\mathbf{a}|\mathbf{s},\mathbf{z})}\left[f(\mathbf{s},\mathbf{a},\mathbf{z})\right]-\mathbb{E}_{n,\phi(\mathbf{z}|n)}\mathbb{E}_{\mathbf{s}\sim\mathcal{D}_n,\mathbf{a}\sim\beta(\mathbf{a}|\mathbf{s},\mathbf{z})}\left[f(\mathbf{s},\mathbf{a},\mathbf{z})\right]-\eta\right).$$

This unconstrained objective implies that if the expected difference in scores of out-of-distribution embeddings and in-distribution embeddings is less than a threshold η, λ is going to be adjusted to 0, on the contrary, λ is likely to take a larger value, used to punish the over-estimated value function. This objective encourages that out-of-distribution embeddings score lower than in-distribution embeddings, thus performing embedding inference will not lead to these out-of-distribution embeddings that are falsely and over-optimistically scored by the learned score model.

In our experiments, we tried five different values for the Lagrange threshold η (1.0, 2.0, 3.0, 4.0, and 5.0). We did not observe a significant difference in performance across these values. Therefore, we simply set $\eta = 2.0$.

Hyper-parameters. In Table 3, we provide the hyper-parameters of the task embedding $\phi(\mathbf{z}|\mathbf{s})$, the contextual behavior policy $\beta(\mathbf{a}|\mathbf{s},\mathbf{z})$, and the score function $f(\mathbf{s},\mathbf{a},\mathbf{z})$. For the gradient ascent update steps (used for embedding inference), we set $K = 100$ for all the embedding inference rules in experiments.

In Table 5, we provide the number of sub-tasks, the number of trajectories in each sub-task, and the dimension of the embedding for each sub-task (behavior policy). The selection of hyperparameter N is based on two evaluation metrics: (1) the fitting loss of the decomposed behavioral policies to the offline data, and (2) the testing performance of DROP. Specifically,

- (Step1) Over a hyperparameter (the number of sub-tasks) set, we conduct the hyperparameter search using the fitting loss of behavior policies, then we choose/filter the four best hyperparameters;
- (Step2) We follow the normal practice of hyperparameter selection and tune the four hypermeters selected in Step1 by interacting with the simulator to estimate the performance of DROP under each hyperparameter setting.

Algorithm 1 DROP: Online fine-tuning (checkpoint-level)

Require: Env, last T checkpoint models: $\beta_t(\mathbf{a}|\mathbf{s},\mathbf{z})$ and $f_t(\mathbf{s},\mathbf{a},\mathbf{z})$ $(t=1,\cdots,T)$.
$R_{\text{MAX}} = -\infty$.
$\beta_{\text{best}} \leftarrow$ None.
$f_{\text{best}} \leftarrow$ None.
while $t = 1,\cdots,T$ **do**
  $\mathbf{s}_0$ = Env.Reset().
  $\mathbf{z}_0^*(\mathbf{s}_0) \leftarrow$ Conduct embedding inference with DROP-Best.
  Return $\leftarrow$ Evaluate β_t and f_t on Env, setting $\pi^*(\mathbf{a}|\mathbf{s}) = \beta(\mathbf{a}|\mathbf{s},\mathbf{z}_0^*(\mathbf{s}_0))$.
  if $R_{\text{MAX}} <$ Return **then**
    Update the best checkpoint models: $\beta_{\text{best}} \leftarrow \beta_t$, $f_{\text{best}} \leftarrow f_t$.
    Update the optimal return: $R_{\text{MAX}} \leftarrow$ Return.
  end if
end while
Return: β_{best} and f_{best}.

We provide the hyperparameter sets in Table 4. In Step2, we tune the (filtered) hyperparameters using 1 seed, then evaluate the best hyperparameter by training on an additional 4 seeds and finally report the results on the 5 total seeds (see next "evaluation protocol"). In Antmaze domain, a single fixed N works well for many tasks; while in Gym-mujoco and Adroit domains, we did not find a fixed N that provides good results for all tasks in the corresponding domain in D4RL, thus we use the above hyperparameter selection rules (Step1 and Step2) to choose the number.

Table 4. Hyperparameter (the number of sub-tasks) set.

tasks	the number of sub-tasks
Antmaze	500 (v0), 150 (v2)
Gym-mujoco	10, 20, 50, 100, 200, 500, 800, 1000
Adroit	10, 20, 50, 100, 200, 500, 800, 1000

Baseline details. For the comparison of our method to prior iterative offline RL methods, we consider the v0 versions of the datasets in D4RL[3]. We take the baseline results of BEAR, BCQ, CQL, and BRAC-p from the D4RL paper (Fu et al., 2020), and take the results of TD3+BC from their origin paper (Fujimoto and Gu, 2021). For the comparison of our method to prior non-iterative offline RL method, we use the v2 versions of the dataset in D4RL. All the baseline results of behavior cloning (BC), Decision Transform (DT), RvS-R, and Onestep are taken from Emmons et al. (2021). In our implementation of COMs, we take the parameters (neural network weights) of behavior policies as the design input for the score model; and during testing, we conduct parameters inference (outer-level optimization) with 200 steps gradient ascent over the learned score function, then the rollout policy is initialized with the inferred parameters. For the specific architecture, we instantiate the policy network with $\dim(\mathcal{S})$ input units, two layers with 64 hidden units, and a final output layer with $\dim(\mathcal{A})$.

Evaluation protocol. We evaluate our results over 5 seeds. For each seed, instead of taking the final checkpoint model produced by a training loop, we take the last T (T = 6 in our experiments) checkpoint models, and evaluate them over 10 episodes for each checkpoint. That is to say, we report the average of the evaluation scores over $5_{\text{seed}} \times 6_{\text{checkpoint}} \times 10_{\text{episode}}$ rollouts.

Online fine-tuning (checkpoint-level): Instead of re-training the learned (final) policy with online rollouts, we fine-tune our policy with enumerated trial-and-error over the last T checkpoint models (Algorithm 1). Specifically, for each seed, we run the last T checkpoint models in the environment over one episode for each checkpoint. The checkpoint model which achieves the maximum episode return is returned. In essence, this fine-tuning procedure imitates the online RL evaluation protocol: if the current policy is unsatisfactory, we can use checkpoints of previous iterations of the policy.

Online fine-tuning (embedding-level): The embedding-level fine-tuning aims to find a suitable gradient ascent step that is used to conduct the embedding inference in DROP-Grad *or* DROP-Grad-Ada. Thus, we enumerate a list of gradient update steps and pick the best update step (according to the episode returns).

Codebase. Our code is based on d3rlpy: `https://github.com/takuseno/d3rlpy`. We provide our source code in the supplementary material.

[3]We noticed that Maze2D-v0 in the D4RL dataset (https://rail.eecs.berkeley.edu/datasets/) is not available, so we used v1 version instead in our experiment. For simplicity, we still use v0 in the paper exposition.

Algorithm 2 DROP: Online fine-tuning (embedding-level)

Require: Env, last T checkpoint models: $\beta_t(\mathbf{a}|\mathbf{s}, \mathbf{z})$ and $f_t(\mathbf{s}, \mathbf{a}, \mathbf{z})$ $(t = 1, \cdots, T)$.

$R_{\text{MAX}} = -\infty$.
$\beta_{\text{best}} \leftarrow$ None.
$f_{\text{best}} \leftarrow$ None.
$k_{\text{best}} \leftarrow 0$.
while $t = 1, \cdots, T$ **do**
  while $k = 1, \cdots, K_{\text{max}}$ **do**
    $\mathbf{s}_0$ = Env.Reset().
    # Conduct embedding inference with DROP-Grad *or* DROP-Grad-Ada
    Return $\leftarrow$ Evaluate β_t and f_t on Env, setting $\pi^*(\mathbf{a}|\mathbf{s}) = \beta(\mathbf{a}|\mathbf{s}, \mathbf{z}^*(\mathbf{s}_0))$ *or* $\beta(\mathbf{a}|\mathbf{s}, \mathbf{z}^*(\mathbf{s}))$, where we conduct k gradient ascent steps to obtain $\mathbf{z}^*(\mathbf{s}_0)$ *or* $\mathbf{z}^*(\mathbf{s})$.
    if $R_{\text{MAX}} <$ Return **then**
      Update the best checkpoint models: $\beta_{\text{best}} \leftarrow \beta_t$, $f_{\text{best}} \leftarrow f_t$.
      Update the best gradient update step: $k_{\text{best}} \leftarrow k$.
      Update the optimal return: $R_{\text{MAX}} \leftarrow$ Return.
    end if
  end while
end while

Return: β_{best}, f_{best} and k_{best}.

Computational resources. The experiments were run on a computational cluster with 22x GeForce RTX 2080 Ti, and 4x NVIDIA Tesla V100 32GB for 20 days.

D. Additional Results

Comparison with iterative offline RL baselines. Here, we compare the performance of DROP (Grad, Best-Ada, and Grad-Ada) to iterative offline RL baselines (BEAR (Kumar et al., 2019), BCQ (Fujimoto et al., 2019), CQL (Kumar et al., 2020), BRAC-p (Wu et al., 2019), and TD3+BC (Fujimoto and Gu, 2021)) that perform iterative bi-level offline RL paradigm with (explicit or implicit) value/policy regularization in inner-level. In Table 6, we present the results for AntMaze, Gym-MuJoCo, and Adroit suites in standard D4RL benchmark (*-v0), where we can find that DROP-Grad-Ada performs comparably or surpasses prior iterative bi-level works on most tasks: outperforming (or comparing) these policy regularized methods (BRAC-p and TD3+BC) on 25 out of 33 tasks and outperforming (or comparing) these value regularized algorithms (BEAR, BCQ, and CQL) on 19 out of 33 tasks.

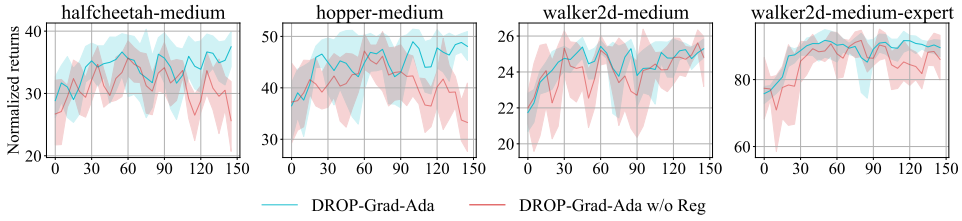

Figure 6. The performance comparison of DROP-Grad-Ada and DROP-Grad-Ada w/o Reg, where we ablate the conservative regularization for the *w/o Reg* implementation. The y-axis denotes the normalized return, the x-axis denotes the number of gradient-ascent steps used for embedding inference at deployment.

Ablation studies. In Figure 6, we provide more results for the ablation of the conservative regularization term in Equation 8 in the main paper. We can find that for the halfcheetah-medium and hopper-medium tasks, the performance of DROP-Grad-Ada w/o Reg depends on the choice of the gradient update steps, showing that too small or too large number of gradient update step deteriorates the performance. Such a result is also consistent with COMs (Trabucco et al., 2021), which also observes the sensitivity of naive gradient update (*i.e.*, *w/o Reg*) to the number of update steps used for design input inference. By comparison, the conservative score model learned with DROP-Grad-Ada exhibits more stable and robust performance to the gradient update steps.

Further, we also find that in walker2d-medium and walker2d-medium-expert tasks, the naive gradient update (*w/o Reg*) does not affect performance significantly across a wide range of gradient update steps. The main reason is that although the

Table 5. The number (N) of sub-tasks, the number (M) of trajectories in each sub-task, and the dimension ($\dim(\mathbf{z})$) of the embedding for each sub-task.

Domain	Task Name	Parameters (*-v0)			Parameters (*-v2)		
		N	M	$\dim(\mathbf{z})$	N	M	$\dim(\mathbf{z})$
Maze 2D	umaze	500	5	5			
	medium	150	50	5			
	large	100	15	5			
Antmaze	umaze	500	50	5	150	50	5
	umaze-diverse	500	50	5	150	50	5
	Medium-play	500	50	5	150	50	5
	Medium-diverse	500	50	5	150	50	5
	Large-play	500	50	5	150	50	5
	Large-diverse	500	50	5	150	50	5
halfcheetah	random	1000	1	5	1000	1	5
	medium	100	2	5	100	2	5
	medium-expert	1000	1	5	1000	1	5
	medium-replay	50	10	5	50	10	5
hopper	random	100	2	5	100	2	5
	medium	100	5	5	100	5	5
	medium-expert	100	2	5	100	2	5
	medium-replay	50	5	5	10	30	5
walker2d	random	500	2	5	500	2	5
	medium	50	5	5	50	5	5
	medium-expert	50	5	5	50	5	5
	medium-replay	1000	5	5	10	50	5
door	cloned	1000	2	5			
	expert	500	5	5			
	human	50	3	5			
hammer	cloned	1000	1	5			
	expert	500	5	5			
	human	20	3	5			
pen	cloned	500	5	5			
	expert	500	5	5			
	human	50	5	5			
relocate	cloned	500	5	5			
	expert	500	5	5			
	human	50	4	5			

excessive gradient updates lead to faraway embeddings, conditioned on the inferred embeddings, the learned contextual behavior policy can safeguard against the embeddings distribution shift. Compared to prior model-based optimization that conducts direct gradient optimization (inference) over the design input itself, such "self-safeguard" is a special merit in the offline RL domain as long as we reframe the offline RL problem as one of model-based optimization and conduct inference over the embedding space. Thus, we encourage the research community to pursue further into this model-based optimization view for the offline RL problem.

Table 6. Comparison of our method to prior offline methods that perform iterative (regularized) RL paradigm on D4RL. We take the baseline results of BEAR, BCQ, CQL, and BRAC-p from Fu et al. (2020), and the results of TD3-BC from Fujimoto and Gu (2021). For all results of our method (DROP), we average the normalized returns across 5 seeds; for each seed, we run 10 evaluation episodes. For proper comparison, we use ▲ and ▲ to denote DROP (*-Ada) achieves *comparable or better* performance compared with value and policy regularized offline RL methods respectively.

	Task Name	Value Reg.			Policy Reg.		DROP-		
		BEAR	BCQ	CQL	BRAC-p	TD3+BC	Grad	Best-Ada	Grad-Ada
antmaze	umaze	73.0	78.9	74.0	50.0	–	72.0	78.0▲▲	80.0▲▲
	umaze-diverse	61.0	55.0	84.0	40.0	–	48.0	62.0 ▲	66.0 ▲
	medium-play	0.0	0.0	61.2	0.0	–	24.0	34.0 ▲	30.0 ▲
	medium-diverse	8.0	0.0	53.7	0.0	–	20.0	24.0 ▲	30.0 ▲
	large-play	0.0	6.7	15.8	0.0	–	24.0	36.0▲▲	42.0▲▲
	large-diverse	0.0	2.2	14.9	0.0	–	14.0	20.0▲▲	26.0▲▲
halfcheetah	random	25.1	2.2	35.4	24.1	10.2	2.3	2.3	2.3
	medium	41.7	40.7	44.4	43.8	42.8	42.4	42.9▲▲	43.1▲▲
	medium-expert	53.4	64.7	62.4	44.2	97.9	86.6	88.5▲	88.9▲
	medium-replay	38.6	38.2	46.2	45.4	43.3	39.5	40.4	40.3
hopper	random	11.4	10.6	10.8	11.0	11.0	5.1	5.4	5.5
	medium	52.1	54.5	58.0	32.7	99.5	57.5	60.3▲	59.5▲
	medium-expert	96.3	110.9	98.7	1.9	112.2	103.5	102.5	105.9▲
	medium-replay	33.7	33.1	48.6	0.6	31.4	48.0	83.4▲▲	87.4▲▲
walker2d	random	7.3	4.9	7.0	-0.2	1.4	2.8	3.0 ▲	3.0 ▲
	medium	59.1	53.1	79.2	77.5	79.7	76.5	75.8▲▲	79.1▲▲
	medium-expert	40.1	57.5	111.0	76.9	101.1	107.5	106.8▲▲	106.9▲▲
	medium-replay	19.2	15.0	26.7	-0.3	25.2	37.4	60.9▲▲	61.9▲▲
door	cloned	-0.1	0.0	0.4	-0.1	–	0.5	2.5	2.7▲▲
	expert	103.4	99.0	101.5	-0.3	–	98.6	102.2▲▲	102.6▲▲
	human	-0.3	0.0	9.9	-0.3	–	3.3	1.9 ▲	3.0 ▲
hammer	cloned	0.3	0.4	2.1	0.3	–	0.3	0.3 ▲	0.3 ▲
	expert	127.3	107.2	86.7	0.3	–	65.7	73.3 ▲	77.7 ▲
	human	0.3	0.5	4.4	0.3	–	1.1	0.3	2.1 ▲
pen	cloned	26.5	44.0	39.2	1.6	–	76.7	77.1▲▲	82.4▲▲
	expert	105.9	114.9	107.0	-3.5	–	113.1	118.6▲▲	116.7▲▲
	human	-1.0	68.9	37.5	8.1	–	71.1	85.2▲▲	81.5▲▲
relocate	cloned	-0.3	-0.3	-0.1	-0.3	–	0.1	0.5▲▲	0.2▲▲
	expert	98.6	41.6	95.0	-0.3	–	2.5	6.2 ▲	5.4 ▲
	human	-0.3	-0.1	0.2	-0.3	–	0.0	0.0	0.0

References

Xi Chen, Ali Ghadirzadeh, Tianhe Yu, Yuan Gao, Jianhao Wang, Wenzhe Li, Bin Liang, Chelsea Finn, and Chongjie Zhang. Latent-variable advantage-weighted policy optimization for offline rl. *arXiv preprint arXiv:2203.08949*, 2022.

Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. Rvs: What is essential for offline rl via supervised learning? *arXiv preprint arXiv:2112.10751*, 2021.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: datasets for deep data-driven reinforcement learning. *CoRR*, abs/2004.07219, 2020. URL https://arxiv.org/abs/2004.07219.

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

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

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

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

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

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

Brandon Trabucco, Aviral Kumar, Xinyang Geng, and Sergey Levine. Conservative objective models for effective offline model-based optimization. In *International Conference on Machine Learning*, pages 10358–10368. PMLR, 2021.

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

Wenxuan Zhou, Sujay Bajracharya, and David Held. Plas: Latent action space for offline reinforcement learning. *arXiv preprint arXiv:2011.07213*, 2020.