A Remarks on Test-time Policy Improvement

In this section, we remark on how ACT and DT improve their policies with the offline dataset. The training phase of ACT and DT can be interpreted as memorizing the offline data together with its hindsight information, either RTG or the advantages. DT and ACT improve the policy by conditionally generating an above-average action during the testing phase, which sets them apart from traditional RL algorithms which optimize policies during training. Such test-time improvement is done by retrieving actions with the desired hindsight information. In DT, the retrieval is accomplished by providing DT with a slightly over-estimated RTG. In ACT, the relationship between advantages and policy performance enables us to make the action generation grounded on theory.

Recall the Performance Difference Lemma:

Lemma A.1. *(Performance Difference Lemma (Kakade and Langford 2002)) Consider an infinite horizon MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, r, \rho_0, \gamma \rangle$. We have*

$$\eta(\pi) - \eta(\pi') = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi'}(s_t, a_t) \right].$$

In ACT, we train the value functions via expectile regression, which smoothly interpolates between estimating (Q^β, V^β) and (Q^*_β, V^*_β). We take the former one as an example, which corresponds to setting $\sigma_1 = 0.5$. In such case, $\pi' = \beta$. Let $\epsilon = \max_{s,a} |\hat{A}^\beta(s,a) - A^\beta(s,a)|$ be the maximum error between the estimated advantage $\hat{A}^\beta$ and the ground truth A^β. Then, we have

$$\begin{aligned}
&\eta(\pi) - \eta(\beta) \\
&= \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t A^\beta(s_t, a_t) \right] \\
&= \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t \hat{A}^\beta(s_t, a_t) \right] \\
&\quad + \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t (A^\beta(s_t, a_t) - \hat{A}^\beta(s_t, a_t)) \right] \\
&\geq \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t \hat{A}^\beta(s_t, a_t) \right] - \frac{\epsilon}{1-\gamma}.
\end{aligned}$$

We denote as Δ_c the resulting performance difference lower bound $\mathbb{E}_{\tau \sim \pi}[\sum_{t=0}^{\infty} \gamma^t \hat{A}^\beta(s_t, a_t)] - \frac{\epsilon}{1-\gamma}$. An intuitive interpretation of the above mathematics is that, if during testing we consistently provide ACT with a positive target advantage, we will improve the performance over the behavior policy β by a margin of Δ_c.

However, Δ_c is not guaranteed to be positive since it is also coupled with the error ϵ. The advantage estimation can be very poor in the limit of $\sigma_1 \to 1$. We do observe that when setting $\sigma_1 = 0.7$, the performance of ACT drops slightly in some tasks that we are concerned about. We speculate that the drop can be explained by the mismatch in the data distribution, since when $\sigma_1 = 0.7$ the learned policy deviates from the behavior policy of the offline data. Besides, in foreign states that ACT is not trained on, it may not be able to generate actions that match the desired advantages. Similar dilemma also bothers policy gradient RL algorithms (Schulman et al. 2015, 2017). It is interesting to apply the idea of trust-region (Schulman et al. 2015) to test-time policy improvement, and we leave it for future research.

B Implementation Details

B.1 Architectures and Hyper-parameters of ACT

In this section, we provide a detailed description of the implementations of ACT as well as other baseline methods. The hyper-parameters for ACT, which are shared across all of the evaluation tasks, are listed in Table 3. For hyper-parameters that vary for different tasks, we will elaborate on them in the corresponding subsections. It is worth noting that for most of the parameters, we directly inherit their value from existing literature without tuning.

Table 3: Hyper-parameters of ACT

Architecture	Hyper-parameters	Value
Q_θ and V_ψ	# of hidden layers	3
	Layer width	256
	Learning rate	3×10^{-4}
	Discount γ	0.99
	Polyak coeff.	0.005
	Batch size	256
	Training steps	200k
	Expectile σ_1	0.5 or 0.7
c_ϕ	# of hidden layers	3
	Layer width	256
	Learning rate	3×10^{-4}
	Batch size	256
	Training steps	500k
	Expectile σ_2	0.98
	Weight decay	5×10^{-4}
ACT	# of encoder layers	3
	# of decoder layers	3
	# of attention heads	1
	Embedding dimension	128
	Nonlinearity function	ReLU
	Batch size	64
	Context length	20
	Dropout	0.1
	Learning rate	10^{-4}
	Lr Warmup steps	10000
	Grad norm clip	0.25
	Weight decay	10^{-4}
	Adam betas	[0.9, 0.999]
	Training steps	500k or 1M

Value functions. We parameterize the value functions Q_θ and V_ψ as 3-layer MLPs respectively, and use Equation 1

to iteratively optimize the value functions. To overcome the over-estimation issue (Fujimoto, van Hoof, and Meger 2018), we maintain two independent value networks $\hat{V}_{\psi_1}$ and $\hat{V}_{\psi_2}$ and use the minimum of their outputs for bootstrapping. We also introduce target networks $\hat{V}_{\bar{\psi}_1}, \hat{V}_{\bar{\psi}_2}$ to give target values for stability. The parameter $\bar{\psi}_i$ is updated in a moving average manner, i.e., $\bar{\psi}_i^{k+1} = (1-\alpha)\bar{\psi}_i^k + \alpha\psi_i^{k+1}$, with the coefficient α being 0.005 across all tasks.

Predictor network c_ϕ. The condition network is also parameterized as a 3-layer MLP, and optimized toward the 0.98 expectile of the advantage.

ACT. The encoder part of ACT consists of three attention layers, each of which is composed of a self-attention module and a feed-forward network. The attention part uses the causal mask to mask out future tokens to prevent information leaks. For positional encoding, we use sinusoidal positional encoding which is proposed in the original Transformer (Vaswani et al. 2017).

The decoder part is adapted from the Transformer decoder. Specifically, we feed a sequence of target advantages into the decoder layer and apply 1) a diagonal mask on the advantage sequence, so that advantages at other timesteps are masked out; and 2) a causal mask on the output sequence of the encoder, so as to prevent information leak from the future. In this way, we are able to implement the core component of ACT within the Transformer architecture.

Finally, we further project the output from the decoder with a linear layer and obtain the predicted deterministic actions.

B.2 Details about Experiments on Deterministic Gym MuJoCo tasks

We provide details about ACT and the baseline methods involved in our evaluation.

Environments and datasets. We use Gym MuJoCo, which includes a wide range of deterministic and continuous control tasks to assess the performance of ACT as well as the baseline methods. For the offline dataset, we choose three levels of quality: 1) *medium*, which is collected by a pretrain SAC (Haarnoja et al. 2018) policy that achieves 30% of the expert performance; 2) *medium-replay*, which consists of the data in the replay buffer of a medium-level policy; 3) *medium-expert*, which is a half-half mixture of data collected by a medium-level policy and an expert-level policy. We use the *-v2* version of datasets.

ACT. On this deterministic benchmark, we sweep the value of σ_1 in $\{0.5, 0.7\}$ and use GAE with $\lambda = 0$. GAE(0) has lower bias compared to IAE since it gets rid of the approximated state-action value function $\hat{Q}_\theta$ while not introducing additional variance due to determinism of the environment. During the course of training, we found that the inherent instability of TD training can lead to occasional oscillations in the value function. Therefore, for this benchmark, we additionally introduce a stage of model selection to assist us in identifying well-fitted models. Specifically, we train in parallel a group of value functions $\{\hat{V}_{\psi_i}\}_{i=1}^n$, each of which is trained with distinct data batches and bootstraps independently. After the training is done, we use the one with the lowest TD error on the whole offline dataset, measured by $\frac{1}{|\mathcal{D}|}\sum_j \mathcal{L}_{\sigma_1}(r_j + \gamma\hat{V}_{\psi_i}(s'_j) - \hat{V}_{\psi_i}(s_j))$, to label the offline dataset and proceed with subsequent steps. The rationale behind this is that lower error on the offline dataset means better alignment with the observed rewards. Recent work (Li et al. 2023) also reveals that, lower TD error on the validation set is a positive signal for higher performance. In our setting, the validation/test set is exactly the offline dataset since we are actually using the value functions to label the offline datasets.

The choice of σ_1 for each task is listed in Table 4. For most of the datasets, $\sigma_1 = 0.7$ works the best since it provides action evaluation from the perspective of an improved policy. For some tasks, $\sigma_1 = 0.5$ works the best, probably due to the reasons discussed in Appendix A.

Table 4: Choices of σ_1 for each dataset.

Dataset	σ_1
halfcheetah-medium	0.7
halfcheetah-medium-replay	0.7
halfcheetah-medium-expert	0.7
hopper-medium	0.5
hopper-medium-replay	0.7
hopper-medium-expert	0.5
walker2d-medium	0.5
walker2d-medium-replay	0.5
walker2d-medium-expert	0.7

Decision Transformer. To make a fair comparison, we reimplemented the Decision Transformer according to the code provided at the official repository and modified the transformer layers of DT to 6 to match the parameter count of ACT. For the initial RTG during test time, we reused the RTGs in the authors' choice, i.e., $\{6000, 12000\}$ for *halfcheetah*, $\{1800, 3600\}$ for *hopper*, and $\{2500, 5000\}$ for *walker2d*, and selected the higher score as the result.

B.3 Details about the 2048 Game

Environment and datasets. The 2048 game is a 4×4 board game. In the beginning, some of the grids are populated with randomly placed 2 or 4. Each time the player can choose a direction and the tiles on the board are moved along the direction. The tiles with identical values are combined into one tile with double the values. The goal of this environment is to reach a tile of 128. The episode will be ended when the tile of 128 is produced and a reward of 1 will be correspondingly assigned. Thus, the maximal possible return is exactly 1. We use the existing implementation and dataset of this game from (Paster, McIlraith, and Ba 2022). The dataset consists of 5M steps of data which are collected by a mixture of a random policy and an expert policy trained with PPO (Schulman et al. 2017).

We list the distribution of trajectory lengths and returns in Figure 7 and Table 5.

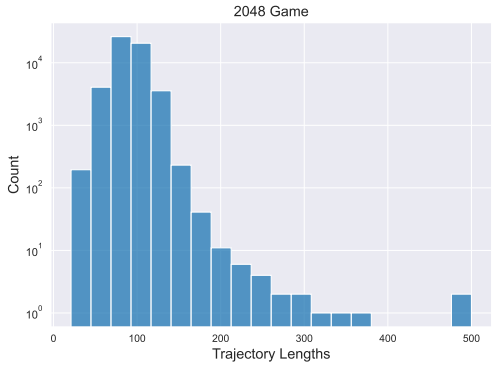

Figure 7: Distribution of the trajectory lengths in the 2048 datasets

Table 5: Distribution of the returns in the 2048 dataset. The return for each trajectory is either 0 or 1, so we list the counts for each type as well as the total number of the trajectories.

Return	# of trajectories
0.0	25741
1.0	29085
Sum	54826

ACT. In this experiment, we use IAE to account for the stochasticity of the environment when assessing the actions. For σ_1, it is set to 0.5, because $\sigma_1 = 0.5$ means we are improving the policy on the basis of the behavior policy, thus isolating the benefits that come from trajectory stitching and the consideration of stochasticity. Moreover, we take the average rather than the minimum of the double V-value functions since the over-estimation issue is not severe as we are performing on-policy value estimation.

Finally, this game is a discrete control task, so we modified the output layer of ACT to predict the logits of a categorical distribution. The training objective for ACT correspondingly transforms to maximizing the log-likelihood of the ground-truth action.

Decision Transformer. We make a similar modification to DT to adapt it for discrete control. For 2048 game, we set the initial RTG during test time as 1.0, since 1.0 is a known upper bound of the returns in this environment.

B.4 Details about the Stochastic Gym MuJoCo Tasks

Environments and datasets. The original Gym MuJoCo tasks are deterministic both in terms of state transition and reward mechanism. To fulfill the stochasticity, we follow the practice of Yang et al. (2023) to add noise to the action chosen by the policy before it is passed to the simulator. Thus from the perspective of the agent, the outcome from the environment becomes unpredictable, which simulates stochasticity. The noise added to the action is a zero-meaned Gaussian noise $\epsilon \sim \mathcal{N}(0, 1 - \exp(-0.01t) \cdot \omega \sin(t))$. For all tasks, ω is set to 0.1. We train a SAC policy in the stochastic version of the environments for 300k steps and log the replay buffer as the offline datasets.

The distributions of the trajectory returns in offline datasets are plotted in Figure 8.

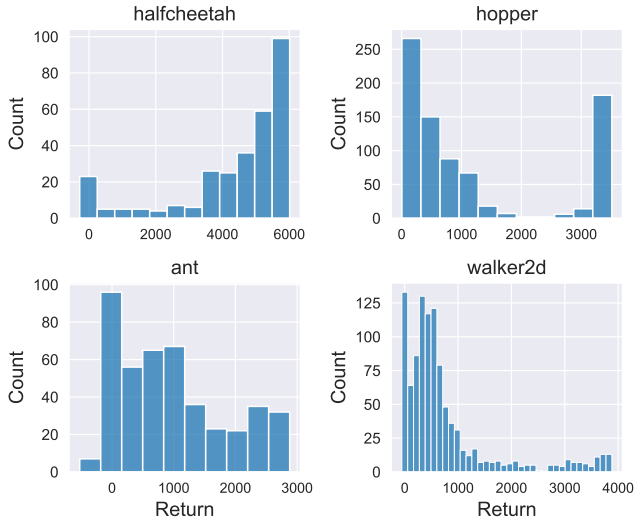

Figure 8: Distributions of the returns in the stochastic Gym MuJoCo datasets.

ACT. The configurations of ACT are the same as those in the 2048 game, so we will not get into the details again here. It is worthwhile to note that although setting σ_1 to 0.7 rather than 0.5 may further boost the performance, we avoided doing this for the purpose of isolating the benefits that come from stitching trajectories and accounting for the stochasticity.

Decision Transformer. The configurations of DT are mostly the same as those used in the deterministic version of the environments. The only difference lies in the initial RTG. We set the RTG according to Table 6:

Table 6: Choices of $\hat{R}_0$ for stochastic Gym MuJoCo tasks.

Dataset	Initial RTG $\hat{R}_0$
halfcheetah-stochastic	$\{6000, 12000\}$
hopper-stochastic	$\{1800, 3600\}$
walker2d-stochastic	$\{2500, 5000\}$
ant-stochastic	$\{2000, 4000\}$

For each task, we designate two initial RTGs, whose values are determined based on the distribution of returns in the offline datasets. We report the higher score between the RTG candidates during evaluation.

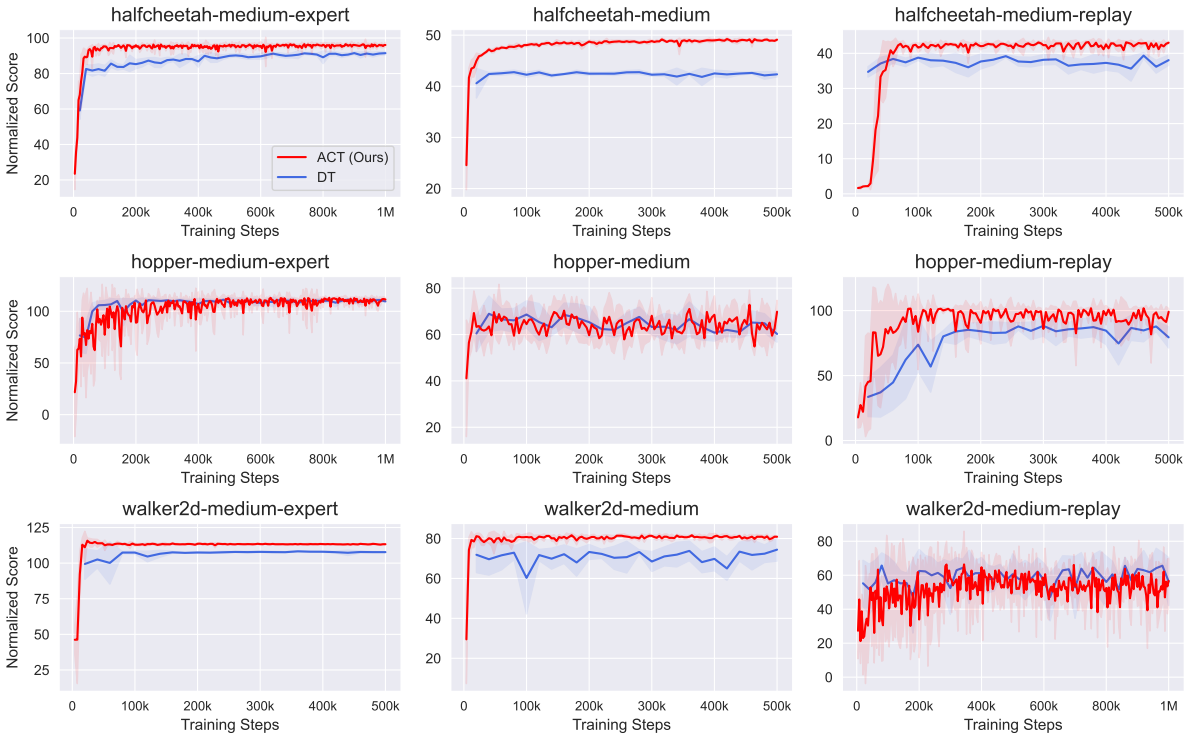

Figure 9: Training curves of DT and ACT on the D4RL datasets. We report the average and one standard deviation across 5 independent runs.

B.5 Details about the Delayed Reward Tasks

Environments and datasets. For this part, we select *hopper* and *walker2d* tasks from Gym MuJoCo, and the *medium* and *medium-expert* datasets from D4RL as the offline datasets. The reason that we omit *halfcheetah* and *medium-replay* is that the datasets may contain timeout trajectories or fragments of the trajectories, which damages the value estimation since the reward is not always given at the terminal states. To simulate the delayed reward setting, we modify the reward at the last timestep to the cumulative rewards of this trajectory and set the past rewards to zeros.

CQL and IQL. We use the implementation from CORL (Tarasov et al. 2022) for IQL and CQL.

Decision Transformer. For DT, we use our own implementation which shares a similar parameter count to our ACT. The initial RTGs are kept the same as the original implementation, as in Table 7.

ACT. For ACT, we set $\sigma_1 = 0.5, \gamma = 1.0$ and GAE with $\lambda = 1.0$. Moreover, we take the average rather than the minimum of the double V-value functions when estimating the values. Other hyper-parameters are kept the same as in Table 3.

C Supplementary Results and Analysis

C.1 Training Curves

In this section, we provide the performance curves of ACT and DT on the D4RL datasets to present the actual dynamics of training. The results are illustrated in Figure 9.

Table 7: Choices of $\hat{R}_0$ for delayed reward tasks.

Dataset	Initial RTG $\hat{R}_0$
hopper	$\{1800, 3600\}$
walker2d	$\{2500, 5000\}$

C.2 Ablations on Effect of σ_2

In Section 6, we conducted an ablation study on the effect of σ_2. In this section, we provide an extended comparison of ACT variants with different values of σ_2, and the results are illustrated in Figure 10. The trend that higher σ_2 leads to improved performance is consistent on all datasets.

C.3 Ablations on Transformer Architecture

Another ablation study focuses on transformer architecture, and in this section, we also provide a detailed comparison with more datasets from D4RL. The results are illustrated in Figure 11. On several datasets `ACT-GPT` suffers from oscillation and inferior performance. Overall the gap between sinusoidal encoding and the learnable positional embedding is minor.

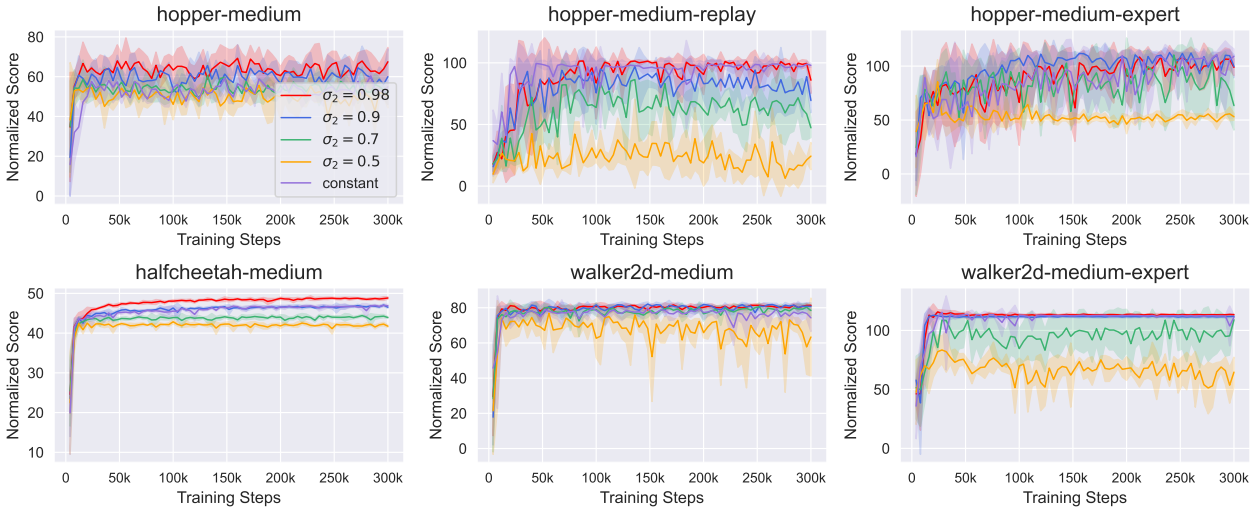

Figure 10: Training curves of ACT with different values of σ_2 and the constant target advantage. We report the average and one standard deviation across 5 independent runs.

Figure 11: Training curves of ACT with different architectural designs. We report the average and one standard deviation across 5 independent runs.

C.4 Running Time Analysis

Throughout the experiments, we evaluate ACT as well as other baseline methods on workstations equipped with NVIDIA RTX A6000 cards. The running time of each method for D4RL datasets is listed in Table 8.

Table 8: Running time for each method.

Algorithm	Running Time
IQL	33min
CQL	2h 51min
DT	1h 04min
ACT	1h 49min

D An Illustrative Example: Cliff-Walking

In this section, we introduce a simplified grid-world environment as a probe to illustrate the actual effect of advantage conditioning. The environment is called *CliffWalking*, as shown in Figure 12.

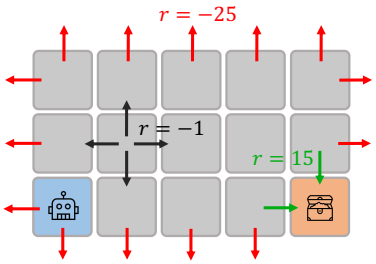

Figure 12: Illustration of the *CliffWalking* environment.

In this environment, the agent is placed at the lower left corner $[0, 0]$ in the beginning. At each timestep, the agent can choose one direction among [`Left`, `Right`, `Up`, `Down`]. If the agent chooses `Left` or `Right`, there is a probability of 0.4 that it actually goes up or down, while for `Up` and `Down`, the action will remain unchanged and will be executed faithfully. If the agent reaches the treasure grid placed at the lower right corner $[4, 0]$, it will receive a reward of 10. If the agent transits out of the grids (falls off the cliff), it will receive a severe punishment of -25. For other scenarios, it will receive a reward of -1 as the cost of the move. Each trajectory will be terminated when the agent reaches the goal state, falls off the cliff, or after 30 moves. The offline dataset, which consists of 10k trajectories, is collected by a random policy.

To reach the grid with treasures, although walking along the lowest row requires the least steps, it also introduces the risk of falling off the cliff. Thus, it is trivial that the optimal strategy in this environment is to first go upward to $[0, 1]$, then walk through the cliff along the middle row to $[4, 1]$, and finally go down to $[4, 0]$. Whenever the optimal agent deviates from the middle row due to the stochasticity of the environment, it should get back to the middle row as soon as possible.

Now, we continue to investigate the policy given by DT and ACT.

Decision Transformer. As a beginning step, we calculate the RTGs for data in the offline dataset and mark the action with the highest RTG in Figure 13. As we can see, RTGs assess the actions by the most successful past experience, i.e., the shortest path toward the goal state. As a result, the final policy produced by DT keeps choosing `Right` (Figure 13) to head for the goal state, although such a decision is risky.

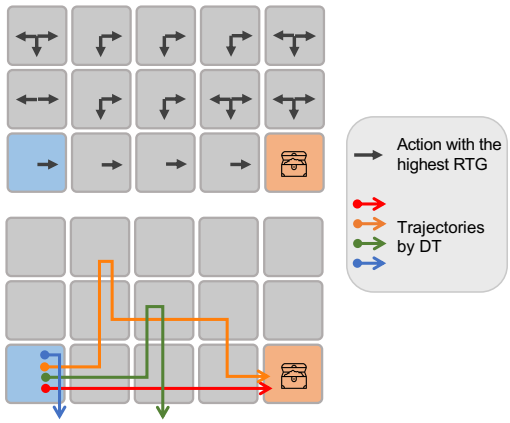

Figure 13: Top: Actions with the highest RTG. Down: Trajectories produced by the trained DT.

ACT with $\sigma_1 = 0.5$. We proceed to investigate the behavior of ACT in this environment. We set $\sigma_1 = 0.5$ and use IAE as the advantage estimator, which means we are using the estimation of on-policy advantage $\hat{A}^\beta$ to assess the actions. Figure 14 depicts the action with the highest advantage in each state and the final trajectories produced by ACT. We find that when $\sigma_1 = 0.5$, ACT fails to produce a valid policy, because the dataset is collected by a random policy and the estimated advantage cannot adequately assess the quality of the actions.

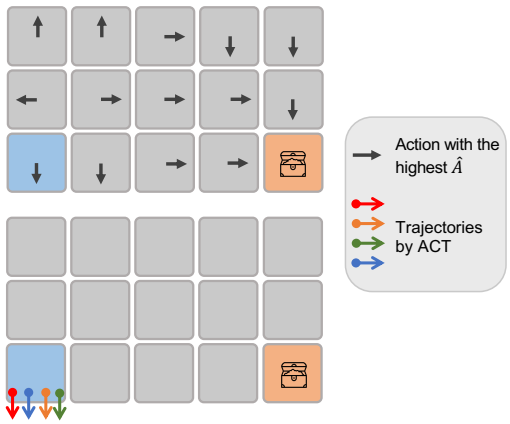

Figure 14: Top: Actions with the highest advantage. Down: Trajectories produced by the trained ACT. The expectile σ_1 is 0.5.

ACT with $\sigma_1 = 0.95$. Finally, we set $\sigma_1 = 0.95$ so that we are approximating the advantage of a nearly optimal in-sample policy. Compared to DT and the above variant, this version of ACT uses the advantage of an already improved policy to assess the quality of actions. The results are depicted in Figure 15. In this version, the advantage suggests the agent first go up to $[0, 1]$, then follow the middle row towards the rightmost column, and finally go down to the goal state. The example trajectories produced by ACT also demonstrate that ACT faithfully executes such optimal strategy. Moreover, whenever it deviates from the middle row, it corrects itself, gets back, and steadily reaches the final goal state in spite of the environmental stochasticity.

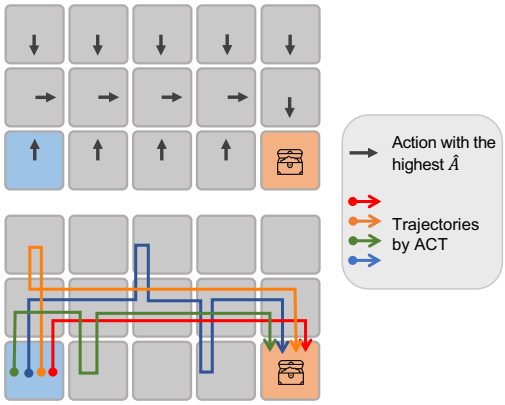

Figure 15: Top: Actions with the highest advantage. Down: Trajectories produced by the trained ACT. The expectile σ_1 is 0.95.

Summarizing the above three experiments, we arrive at two conclusions. First, increasing the value of σ_1 does bring benefits in some circumstances, as higher σ_1 allows us to evaluate the actions from the standpoint of an improved policy. Second, compared with RTG conditioning, advantage conditioning gives a robust assessment based on the return in expectation, rather than occasional random success. Such property provides sequence modeling methods with the possibility of converging to the optimal policy, rather than a blindly optimistic policy as return conditioning will do.