Supplementary Material for Diffused Task-Agnostic Milestone Planner

Anonymous Author(s)
Affiliation
Address
email

A Proof of Proposition 1

Recap We consider a goal-conditioned Markov decision process defined by a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition model $p(s'|s,a)$, and a goal space $\mathcal{G}$. We further consider an encoder $f_\omega : \mathcal{S} \to \mathcal{G}$ which maps a goal state $s_{\text{goal}} \in \mathcal{S}$ into a goal vector $g \in \mathcal{G}$. Goal-conditioned imitation learning is defined by training a goal-conditioned actor $\pi_\psi : \mathcal{S} \times \mathcal{G} \to \mathcal{A}$ to predict the most likely action that a behavior policy $\mu(a|s)$ will take for a given goal $g \in \mathcal{G}$:

$$\text{maximize}_\psi \ \log \mu(a|s,g), \ \text{for } s \in \mathcal{S} \text{ and } a = \pi_\psi(s,g). \tag{1}$$

Using Bayes' rule, $\mu(a|s,g) = \frac{p_\mu(g|s,a)\mu(a|s)}{p_\mu(g|s)}$, where $p_\mu(g|s,a)$ and $p_\mu(g|s)$ are the finite horizon occupancy measures, the objective in (1) can be reformulated as:

$$J(s,g;\psi) := -\log p_\mu(g|s,a) - \log \mu(a|s), \ \text{for } a = \pi_\psi(s,g). \tag{2}$$

In order to estimate the log-likelihood $\log p_\mu(g|s,a)$, we train a discriminative critic function $D_\phi : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to [0,1]$ to distinguish between states that are more likely to come after the current state-action pair, by minimizing the following loss function:

$$\begin{aligned} &J_{\text{critic}}(s,a,s^+,s^-;\phi,\omega) := -\log D_\phi(s,a,f_\omega(s^+)) - \log\left(1 - D_\phi(s,a,f_\omega(s^-))\right), \\ &\text{where, } s \sim p_\mu(s), \ a \sim \mu(a|s), \ s^+ \sim p_\mu(s^+|s,a), \ s^- \sim p_\mu(s^-).\text{`} \end{aligned} \tag{3}$$

We now aim to prove the following proposition:

Proposition A.1. *The optimal critic function D^*_ϕ that minimizes the cross-entropy loss (3) satisfies the following equation for a given goal goal $g = f_\omega(s_{goal})$.*

$$D^*_\phi(s,a,g) = \frac{p_\mu(g|s,a)}{p_\mu(g|s,a) + p_\mu(g)}. \tag{4}$$

Proof. We can rewrite the loss function (3) as,

$$J(s,a;\phi,\omega) = -\sum_{\tilde{g} \in \tilde{\mathcal{G}}_\omega} \left[p_\mu(\tilde{g}|s,a)\log D_\phi(s,a,\tilde{g}) + p_\mu(\tilde{g})\log\left(1 - D_\phi(s,a,\tilde{g})\right)\right], \tag{5}$$

where, $\tilde{\mathcal{G}}_\omega = \{f_\omega(s) | s \in \mathcal{S}\}$. Then, by taking derivative of (5) with respect to $D_\phi(s,a,g)$,

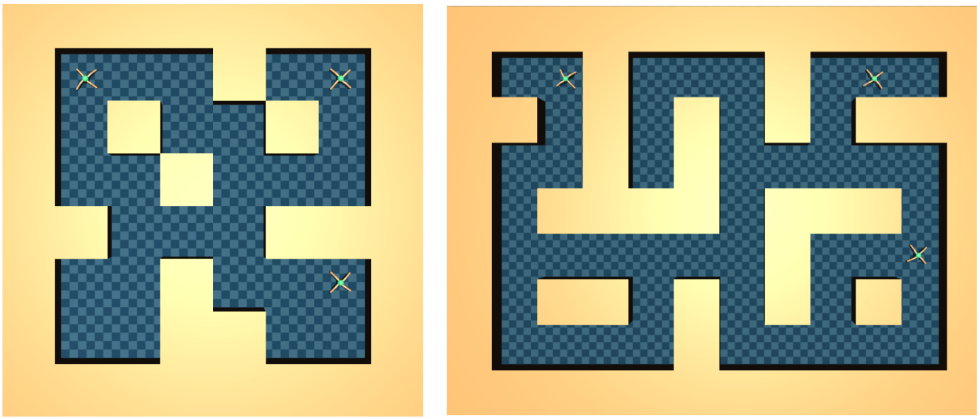
$$\frac{\partial J(s,a;\phi,\omega)}{\partial D_\omega(s,a,g)} = -\frac{p_\mu(g|s,a)}{D_\omega(s,a,g)} + \frac{p_\mu(g)}{1 - D_\omega(s,a,g)}. \tag{6}$$

At $D_\phi(s,a,g) = D^*_\phi(s,a,g)$, the derivative in (6) becomes 0. Therefore,

$$D^*_\phi(s,a,g) = \frac{p_\mu(g|s,a)}{p_\mu(g|s,a) + p_\mu(g)}.$$

□

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.



(a) Antmaze-medium (b) Antmaze-large

Figure 1: Each position marked by an ant illustrates a predefined target position.

B Experiment setting

Multi-goal Antmaze experiments To evaluate DTAMP and baseline methods on multi-goal setting, we predefined three different target positions as shown in Figure 1. During the evaluation, a goal was randomly sampled from the three target positions for each rollout. The models were trained using the same data (`antmaze-medium-play`, `antmaze-medium-diverse`, `antmaze-large-play`, `antmaze-large-diverse`) that was used for single-goal Antmaze experiments.

CALVIN experiments We used the same data split and tasks used in Rosete-Beas et al. [13] to train and evaluate the model in the CALVIN experiment. We set time horizon of a rollout to 300 timesteps when a goal image implies one or two tasks, and 450 timesteps when a goal image implies three tasks.

C Source of baseline performance

D4RL experiments The performances of CQL [11], IQL [10], and ContRL [3] are taken from their corresponding papers. The performance of DT [2] in Antmaze environments is taken from Janner et al. [7]. The performance of DD [1] in Kitchen environment is taken from its corresponding paper. To evaluate DD on Antmaze environments, DD is trained with three different random seeds using the author's code[1], and each trained model is evaluated with 30 rollouts (90 rollouts in total) for each task. While the performance of TT+IQL is presented in Janner et al. [7], it is inaccurate as only 15 rollouts were used for evaluation. For more accurate evaluation, TT and IQL are trained with three different random seeds using the code[2] provided by the author, and each model is evaluated with 30 rollouts (90 rollouts in total) for each task.

Multi-goal Antmaze experiments IQL+HER is implemented based on the author's code[3] by concatenating two-dimensional goal position on the original state vector. We train IQL+HER with three different random seeds and evaluate each model with 150 rollouts. As ContRL already learns a goal-conditioned policy, ContRL is trained with three different random seeds using the author's code[4] without modification. As also TT does not require a goal-specific training, we evaluate TT+IQL using the model trained for single-goal experiments while using the value functions trained by IQL+HER.

CALVIN experiments The performances of CQL+HER, LMP [12], RIL [6], and TACO-RL [13] are taken from Rosete-Beas et al. [13] for the cases that one or two tasks are implied by a goal image. For the case that three tasks are implied by a goal image, we trained each model using the code[5] provided by Rosete-Beas et al. [13], and evaluated each model with 1,000 rollouts.

[1] https://github.com/anuragajay/decision-diffuser/tree/main/code
[2] The author thankfully provided a private github repository
[3] https://github.com/ikostrikov/implicit_q_learning
[4] https://github.com/google-research/google-research/tree/master/contrastive_rl
[5] https://github.com/ErickRosete/tacorl

D Implementation details

Encoder The encoder consists of two neural networks one for each actor and critic. Each neural network has two hidden fully-connected layers with a size of 512, and an output layer. In addition, we normalized each output of the two neural networks, to provide consistent signal-to-noise ratio to the diffusion model during training process.

Actor and critic Each actor and critic has two hidden fully-connected layers with a size of 512, and an output layer. We train five independent critic networks and use the smallest of the values predicted by critics to train actors, as done in Eysenbach et al. [3].

Diffusion model The diffusion model for planning milestones is implemented based on the code provided by Ajay et al. [1], which consists of a temporal U-Net architecture with six residual 1-D convolutional blocks. To let the diffusion model predict milestones $g_{1:K}$ based on an initial state s_0 and goal state s_{goal}, we fixed $g_0 = f_\omega(s_0)$ and $g_{K+1} = f_\omega(s_{\text{goal}})$ during the denoising process, as done in Janner et al. [8].

Image preprocessing Our image preprocessing method used for CALVIN experiment consists of three steps: 1) Resize RGB images size of $200 \times 200 \times 3$ into $128 \times 128 \times 3$. 2) Perform stochastic image shifts of 0-6 pixels. 3) Perform color jitter transform augmentation with a contrast of 0.1, a brightness of 0.1 and hue of 0.02. 4) Normalize value of each pixel to let it falls between -1 and 1.

Visual perception For CALVIN experiments, we utilize a visual perception network consists of three convolutional layers and a spatial softmax layer. It is the same as the perception network used in Lynch et al. [12] and Rosete-Beas et al. [13].

Skill encoder and decoder The skill encoder and decoder are trained using the code provided by Rosete-Beas et al. [13], and have the same architecture as LMP and TACO-RL.

E Training details

D4RL experiments We train the encoder, goal-conditioned actor, critic and diffusion model simultaneously using a unified loss function: $J_{\text{unified}} := J_{\text{actor}} + J_{\text{critic}} + \alpha J_{\text{diffusion}}$, where the coefficient α is fixed to 0.001 for all the experiments. We train the models for 2.5M training steps for Antmaze environments and 1.0M steps for Kitchen environments. We utilize NVIDIA Geforce RTX 3060 Ti graphics card for training our models, taking approximately 14 hours per 1.0M training steps. We use Adam optimizer [9], with a learning rate of 0.0001.

CALVIN experiments We train our model for 1.0M training steps using the same graphics card used for D4RL experiments, taking approximately 40 hours per 1.0M training steps. We use the same optimizer and learning rate used in D4RL experiments.

F Hyperparameter setting

In this section, we describe hyperparameter details.

- We set dimension of goal space $\mathcal{G}$ to 16 for Antmaze environments, and 32 for Kitchen and CALVIN environments.
- We use the number of milestones $K = 30$ with maximum interval $\triangle_{\text{max}} = 32$ for Antmaze environments, and $K = 14$ with maximum interval $\triangle_{\text{max}} = 16$ for Kitchen and CALVIN environments.
- We set λ in Equation (9) in the main paper to be adaptive to scale of estimated log-likelihood $\log p(g|s,a)$, by setting it as $\lambda = \frac{\tilde{\lambda}}{\frac{1}{|\mathcal{B}|}\sum_{(s,a,g)\in\mathcal{B}} |\log p(g|s,a)|}$ as done in Fujimoto and Gu [4], where $\mathcal{B}$ denotes a mini-batch. We use $\tilde{\lambda}$ of 2.5 for Antmaze environments, and 0.05 for Kitchen and CALVIN environments.
- We use diffusion timestep N of 300, diffusion guidance coefficient β of 0.5, and target temporal distance $\triangle_{\text{target}}$ of $0.5\triangle_{\text{max}}$ for the all experiments.
- We use threshold δ of 0.1 to determine whether the agent has reached the targeted milestone.

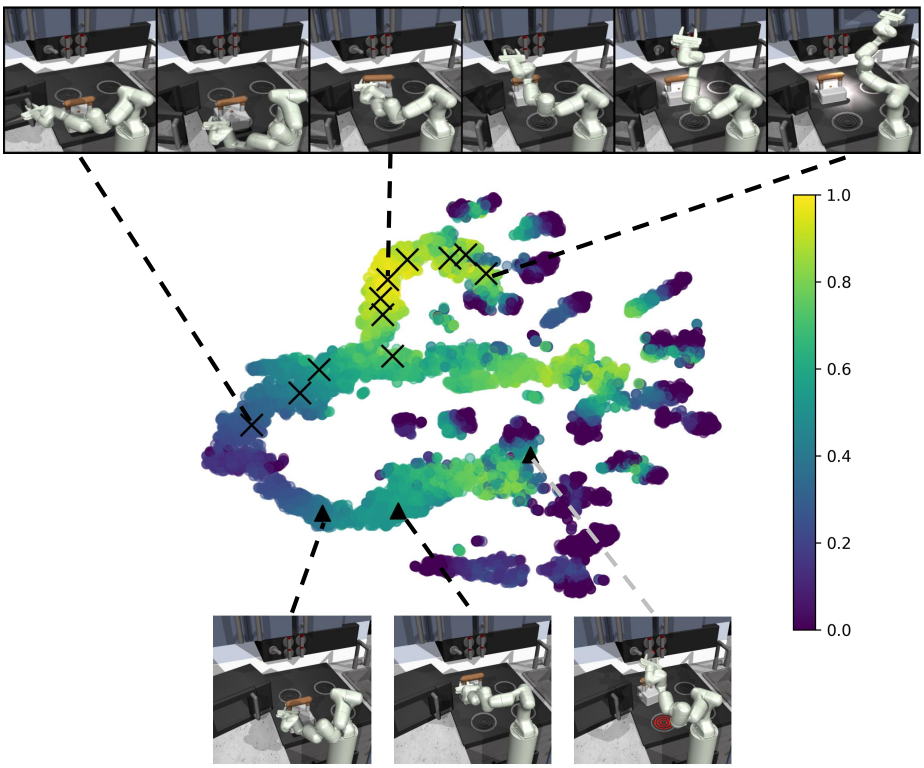


Figure 2: The figure shows t-SNE [15] embedding of the learned latent space. Each data point represents a state in `kitchen-mixed-v0` data. The black x's indicate milestones planned by DTAMP, and the black triangles indicate three states sampled from an arbitrary trajectory. The colors represent state-values estimated using IQL.

G Visualization of the learned latent goal space

In this section, we discuss about the ability of DTAMP to learn a latent goal space that captures the dynamics of an environment. We train the encoder along with the actor and critic, through goal-conditioned imitation learning. By doing so, we make the encoder capture useful information from states, to let the actor and critic distinguish the states and actions that can lead an agent to the goal. There are also similar methods that have been studied through prior work, such as learning forward or inverse dynamics model in a latent space [14, 5] or using contrastive learning based on state occupancy measures [3]. Figure 2 visualizes an example of latent goal space learned by DTAMP using `kitchen-mixed-v0` data. The figure shows that the trained encoder can capture the dynamics of environment, and also let the milestone planner successfully predict milestones in the learned latent space.

H Robustness against environment stochasticity

In order to demonstrate the robustness against environment stocahsticity, we further evaluate DTAMP on `maze2d-umaze-v1` environment of the D4RL benchmark, while adding stochasticity to the transition model. We also compare DTAMP against Diffuser [8], which predicts future states and actions using a diffusion model, and Decision Diffuser (DD) [1], which utilizes an inverse dynamics model to predict actions from the states planned by a diffusion model. In this experiment we let Diffuser and DD predict the whole trajectory only once at the beginning of each rollout as done in DTAMP.

Setting We create three different levels of stochasticity by choice of $p \in \{0, 0.3, 0.5\}$. The environment either executes a random action $a \sim \text{Unif}(\mathcal{A})$ with probability p or executes the action given by the agent with probability $(1 - p)$. Three sets of offline data were collected separately

p	Diffuser	DD	**DTAMP**	Ref. max score
0.0	61.9	92.6	**97.6**	80.5
0.3	29.6	97.0	**99.0**	72.3
0.5	18.5	70.5	**99.4**	64.3

Table 1: Normalized average score with different levels of stochasticity.

according to the choice of p, and each set of data consists of one million transitions collected by a waypoint controller. We also note that the control frequency is reduced threefold by repeating the same action three times in this experiment. By doing so, we reduce the time horizon of the environment (300 timesteps to 100 timesteps) to let Diffuser and DD predict the whole trajectory only once at the beginning of each rollout.

Result The normalized score of each model is shown in Table 1. The column named Ref. max score (reference max score) shows the average score of the waypoint controller, which represents the performance of an optimal agent. We note that even if an agent predicts optimal actions, stochasticity of environment causes inherent performance degradation. In order to neglect this effect and only compare the robustness of each model, the scores shown in Table 1 are normalized respect to the reference max scores (*i.e.*, normalized score $= 100 \times \frac{\text{average score}}{\text{ref. max score}}$).

The result shows that DTAMP is robust against the environment stochasticity and does not show performance degradation in the terms of normalized score. On the other hand, Diffuser shows greater performance degradation as stochasticity increased. DD shows no performance degradation when the stochasticity is small ($p = 0.3$) while showing performance degradation when the stochasiticity is large ($p = 0.5$). The result indicates that our approach using a goal-conditioned actor makes DTAMP robust against environment stochasticity and allows to plan milestones only once for each rollout, while the other diffusion based planners do not.

I Ablation study for CALVIN experiment

We make two modifications for applying DTAMP on image-based manipulation tasks: 1) adding an observation decoder to reconstruct original images from encoded goal vectors 2) using skill-based policy as done in Lynch et al. [12] and Rosete-Beas et al. [13]. To investigate the effect of the modifications, we present ablation studies in this section. We evaluate the average success rates of DTAMP−Decoder which does not use the decoder and DTAMP−LMP which does not use the skill-based policy. In addition, we also evaluate a variation of DTAMP by allowing the agent to re-plan the remaining milestones if the agent has not reached a targeted milestone within a certain time limit[6] (DTAMP+Replanning).

The results shown in Table 2 indicate that not using decoder (DTAMP-Decoder) causes marginal degradation of the performance, while still outperforming the second-best method (TACO-RL [13]). In the meantime, not using skill-based policy (DTAMP-LMP) causes larger degradation of the performance. Furthermore, allowing re-planning results increasing the average success rate by a factor of 1.4.

To take a closer look at the effect of the modifications, we further compare the success rates of the variants of DTAMP for each task (Figure 3). DTAMP−Decoder shows similar success rates to DTAMP for most tasks, while showing a great performance degradation in `turn on/off led` tasks. It is because that `turn on/off led` tasks are designated by the color change of the small LED, which is hard to be captured by the encoder. This result indicates that using decoder to reconstruct the original image helps the encoder to capture the visual features more accurately. In addition, DTAMP−LMP shows marginal degradation of the performance across most tasks. While DTAMP−LMP also shows large performance degradation in `turn on/off led` tasks, is is distinct from the case of DTAMP−Decoder. The agents trained by DTAMP−LMP tend to try to turn the LED on or off by pressing the button, while often failing to move the robot arm to the correct position of the button. From this investigation, we can conclude that the use of decoder helps encoder capture visual features, and the use of skill-based policy makes control more accurate.

[6] We use time limit of 32 timesteps

Number of tasks	TACO-RL	DTAMP	DTAMP − Decoder	DTAMP − LMP	DTAMP + Replanning
1	67.9	82.4	80.2	55.9	87.4
2	27.0	52.4	49.4	24.3	62.6
3	0.4	29.7	23.0	15.2	38.1
Average	31.8	54.8	50.9	31.8	62.7

Table 2: The results of ablation study on the CALVIN benchmark. The values represent success rates in percentage.

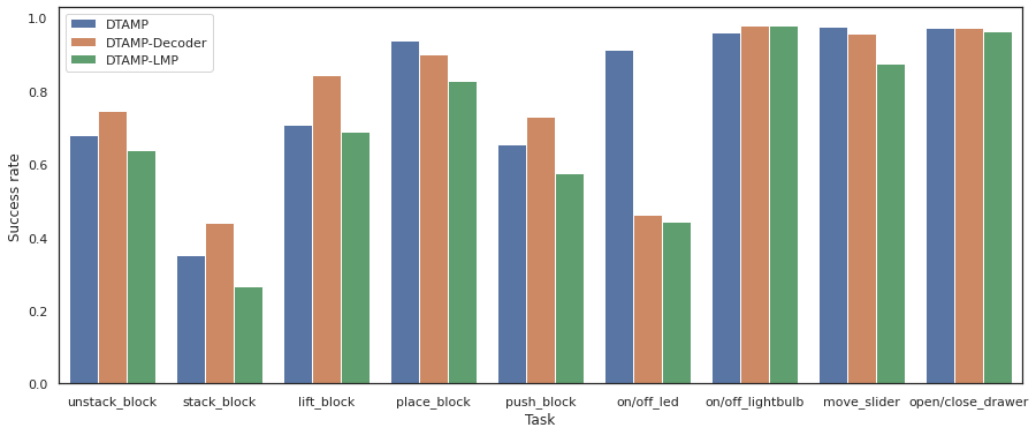


Figure 3: The figure shows the success rate of variants of DTAMP for each task.

References

[1] Anurag Ajay, Abhi Gupta Yilun Du, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making. *arXiv preprint arXiv:2211.15657*, Dec 2022.

[2] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, Virtual conference, Dec 2021.

[3] Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Ruslan Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. In *Advances in Neural Information Processing Systems*, New Orleans, US, Dec 2022.

[4] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems*, Virtual conference, Dec 2021.

[5] Zhaohan Guo, Shantanu Thakoor, Miruna Pîslar, Bernardo Avila Pires, Florent Altché, Corentin Tallec, Alaa Saade, Daniele Calandriello, Jean-Bastien Grill, Yunhao Tang, et al. Byol-explore: Exploration by bootstrapped prediction. In *Advances in neural information processing systems*, New Orleans, US, Dec 2022.

[6] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Proceedings of the Conference on Robot Learning*, Osaka, JP, Oct 2019.

[7] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, Virtual conference, Dec 2021.

[8] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *Proceedings of the International Conference on Machine Learning*, Baltimore, US, Jul 2022.

[9] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations*, San Diego, US, May 2015.

[10] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *Proceedings of the International Conference on Learning Representations*, Virtual conference, Apr 2022.

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

[12] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Proceedings of the Conference on Robot Learning*, Osaka, JP, Oct 2019.

[13] Erick Rosete-Beas, Oier Mees, Gabriel Kalweit, Joschka Boedecker, and Wolfram Burgard. Latent plans for task agnostic offline reinforcement learning. In *Proceedings of the Conference on Robot Learning*, Auckland, NZ, Dec 2022.

[14] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. In *Proceedings of International Conference on Machine Learning*, Stockholm, SE, Jul 2018.

[15] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9:2579–2605, Nov 2008.