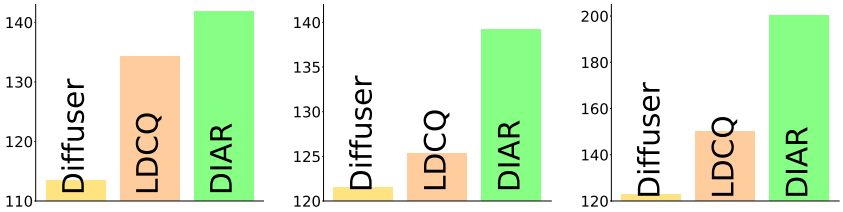
Diffusion-Guided Q-Learning for Offline RL: Adaptive Revaluation for Long-Horizon Decision Making

Jaehyun Park[1], **Yunho Kim**[1], **Sejin Kim**[1], **Byung-Jun Lee**[2], **Sundong Kim**[1]
[1]Gwangju Institute of Science and Technology, [2]Korea University

Abstract:
We introduce DIAR (Diffusion-model-guided Implicit Q-learning with Adaptive Revaluation), a novel offline reinforcement learning approach that tackles learning from fixed datasets and making effective long-horizon decisions. DIAR leverages diffusion models to learn state-action sequence distributions for balanced decision-making combined with value functions. The key innovation is the Adaptive Revaluation mechanism, which dynamically adjusts decision lengths by comparing current and future state values, enhancing long-term flexibility and trajectory selection. DIAR enables precise Q-function learning through diffusion-guided value functions and generates diverse latent trajectories for improved policy robustness. We evaluate DIAR on D4RL benchmarks, demonstrating competitive performance and consistent improvements over existing offline RL methods.

Keywords: Offline RL, Diffusion model, Sparse reward, Long horizon



(a) Maze2D-umaze (b) Maze2D-medium (c) Maze2D-large

Figure 1: Performance comparison across D4RL environments with long-horizon and sparse-reward Maze2D tasks. Our method (DIAR) consistently outperforms Diffuser and LDCQ.

1 Introduction

Offline reinforcement learning (offline RL) is a type of reinforcement learning where an agent learns a policy using pre-collected datasets instead of gathering data through direct interactions with the environment [1]. By avoiding real-world interactions, offline RL eliminates safety concerns and makes efficient use of collected data, which is especially beneficial when gathering new data is costly or time-consuming. However, offline RL depends on the dataset, meaning the policy it learns may perform poorly if the data is low quality or biased. Moreover, a distributional shift can occur during the learning from offline data [2], leading to degraded performance in the real environment.

To overcome these limitations, recent research has leveraged diffusion models, a type of generative model [3]. Incorporating diffusion models allows for learning the overall distribution of state and action spaces, allowing decisions to be made based on this knowledge. Methods such as Diffuser [3] and Decision Diffuser [4] use diffusion models to predict entire decision horizons simul-

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

taneously rather than autoregressively, achieving strong performance in long-horizon tasks. Additionally, methods like LDCQ [5] propose using latent diffusion models to learn the Q-function, allowing the Q-function to make more appropriate predictions for out-of-distribution state-actions. While many diffusion-based offline RL methods bypass Q-functions, recent research has proposed approaches that leverage diffusion models to assist Q-learning [5, 6]. This enables handling Q-values across various states and actions, with diffusion-generated samples improving agent performance.

Therefore, we propose DIAR (Diffusion-model-guided Implicit Q-learning with Adaptive Revaluation), which integrates value functions and diffusion-generated samples into training and decision processes. This approach provides an objective assessment of current states, enabling Q-functions to balance long-horizon decision-making with step-by-step refinement. The Q-function and value function alternately learn from datasets and diffusion samples, while the value function reevaluates decisions to explore optimal action sequences.

DIAR consistently outperforms existing offline RL algorithms, particularly in environments involving complex route planning and long-horizon tasks. Our method achieves strong performance on challenging benchmarks, including Maze2D, AntMaze, and Kitchen [7], demonstrating the potential of diffusion models to enhance policy abstraction and adaptability in offline RL.

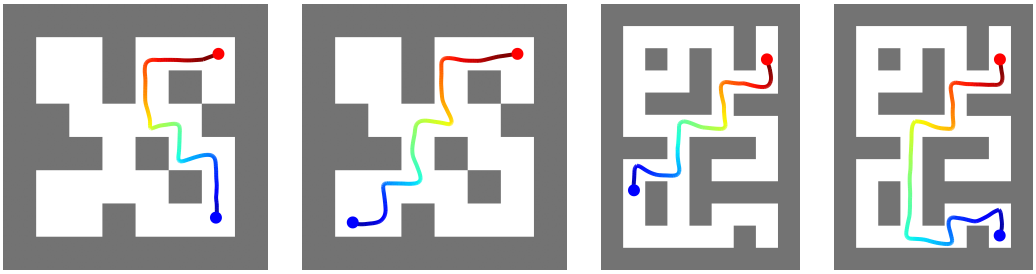


Figure 2: DIAR-generated trajectories in challenging Maze2D situations. DIAR reliably reaches the goal even from starting points (blue) that are far from the goal (red). DIAR shows strong performance regardless of starting position.

2 Preliminary: Latent Diffusion Reinforcement Learning

To train the Q-network, a diffusion model that has trained based on latent representations is required. The first step is to learn how to represent an action-state sequence of length H as a latent vector using β-Variational Autoencoder (β-VAE) [8]. The second step is to train the diffusion model using the latent vectors generated by the encoder of the β-VAE. This allows the diffusion model to learn the latent space corresponding to the action-state sequence. Subsequently, the Q-network is trained using the latent vectors generated by the diffusion model.

Latent representation by β-VAE. The β-VAE plays three key roles in the initial stage of our model training. First, the encoder $q_{\theta_E}(\boldsymbol{z}|\boldsymbol{s}_{t:t+H}, \boldsymbol{a}_{t:t+H})$ must effectively represent the action-state sequence $\boldsymbol{s}_{t:t+H}, \boldsymbol{a}_{t:t+H}$ from the dataset $\mathcal{D}$ into a latent vector $\boldsymbol{z}$. Second, the distribution of $\boldsymbol{z}$ generated by the β-VAE must be conditioned by the state prior $p_{\theta_s}(\boldsymbol{z}|\boldsymbol{s}_t)$. This is learned by minimizing the KL-divergence between the latent vector generated by the encoder and the one generated by the state prior. The formation of the latent vector is controlled by adjusting the β value, which determines the weight of KL-divergence. Lastly, the policy decoder $\pi_{\theta_D}(\boldsymbol{a}_t|\boldsymbol{s}_t, \boldsymbol{z})$ of the β-VAE must be able to accurately decode actions when given the current state and latent vector as inputs. These three objectives are combined to train the β-VAE by maximizing the evidence lower bound (ELBO) [9] as shown in Eq. 1.

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathcal{D}}\Big[\mathbb{E}_{q_{\theta_E}}\Big[\sum_{i=t}^{t+H-1} \log \pi_{\theta_D}(\boldsymbol{a}_i|\boldsymbol{s}_i, \boldsymbol{z})\Big] - \beta D_{KL}(q_{\theta_E}(\boldsymbol{z}|\boldsymbol{s}_{t:t+H}, \boldsymbol{a}_{t:t+H}) \parallel p_{\theta_s}(\boldsymbol{z}|\boldsymbol{s}_t))\Big] \quad (1)$$

Training latent vector with a diffusion model. The latent diffusion model (LDM) effectively learns latent representations, focusing on the latent space instead of the original data samples [10]. The model minimizes a loss function that predict the initial latent $\boldsymbol{z}_t$ generated by the VAE encoder q_ϕ, rather than noise as in traditional diffusion models. H-length trajectory segments $\boldsymbol{s}_{t:t+H}, \boldsymbol{a}_{t:t+H}$ are sampled from dataset $\mathcal{D}$ and paired with initial states and latent variables $(\boldsymbol{s}_t, \boldsymbol{z}_t)$. The focus lies on modeling the prior $p(\boldsymbol{z}|\boldsymbol{s}_t)$ to capture the distribution of latent $\boldsymbol{z}$ given the state $\boldsymbol{s}_t$. A conditional latent diffusion model $\mu_\psi(\boldsymbol{z}|\boldsymbol{s}_t)$ is utilized and refined with a time-dependent denoising function $\mu_\psi(\boldsymbol{z}^j, \boldsymbol{s}_t, j)$ to reconstruct $\boldsymbol{z}^0$ through the denoising step $j \sim [1, T]$. Consequently, the LDM is trained by minimizing the loss function $\mathcal{L}(\psi)$ as given in Eq. 2.

$$\begin{aligned} \mathcal{L}(\psi) &= \mathbb{E}_{j,(\boldsymbol{s},\boldsymbol{a}),\boldsymbol{z}_t,\boldsymbol{z}^j}\left(\|\boldsymbol{z}_t - \mu_\psi(\boldsymbol{z}^j, \boldsymbol{s}_t, j)\|^2\right) \\ j \sim [1,T], (\boldsymbol{s},\boldsymbol{a}) &\sim \mathcal{D}, \boldsymbol{z}_t \sim q_\phi(\boldsymbol{z}|\boldsymbol{s},\boldsymbol{a}), \boldsymbol{z}^j \sim \mu_\psi(\boldsymbol{z}^j|\boldsymbol{z}^0) \end{aligned} \tag{2}$$

Q-network by latent representation. To train the Q-network, Eq. 3 reduces extrapolation errors by restricting policy updates to the empirical distribution of the offline dataset [5]. Prioritizing trajectory accuracy over single-step precision allows the model to mitigate compounding errors and remain adaptable to novel tasks or goals unseen during training. Furthermore, the integration of temporal abstraction and latent space modeling notably enhances the mechanisms underlying credit assignment and improves the effectiveness of policy optimization.

$$Q(\boldsymbol{s}_t, \boldsymbol{z}_t) \leftarrow Q(\boldsymbol{s}_t, \boldsymbol{z}_t) + \alpha \left[r_{t:t+H} + \gamma \max_{\boldsymbol{z}'_{t+H} \sim \mu_\psi} Q(\boldsymbol{s}_{t+H}, \boldsymbol{z}'_{t+H}) - Q(\boldsymbol{s}_t, \boldsymbol{z}_t) \right] \tag{3}$$

The latent vector $\boldsymbol{z}'_{t+H}$ generated by the diffusion model is utilized in the training of the Q-function. The Q-function learns the relation between the $Q(\boldsymbol{s}_{t+H}, \boldsymbol{z}'_{t+H})$ and $Q(\boldsymbol{s}_t, \boldsymbol{z}_t)$ like Eq. 3, which are based on the initial state $\boldsymbol{s}_t$ and latent vector $\boldsymbol{z}_t$ pairs present in the dataset, and the $\boldsymbol{z}'_{t+H}$ generated by the diffusion model. $r_{t:t+H}$ denotes the sum of rewards with discount factor γ. This enables the model to adapt to new tasks or goals that were not observed in the offline data. Furthermore, the integration of temporal abstraction and latent space modeling significantly enhances the mechanism of credit assignment, thereby improving the effectiveness of policy optimization. According to Eq. 3, the trained Q-function is used such that, as shown in Eq. 4, when a state $\boldsymbol{s}_t$ is given, the decision is made by selecting the action that has the highest Q-value.

$$\pi(\boldsymbol{s}_t) = \pi_\theta(\boldsymbol{a}_t | \operatorname*{argmax}_{\boldsymbol{z}_i \sim \mu_\psi(\boldsymbol{z}|\boldsymbol{s}_t)} Q(\boldsymbol{s}_t, \boldsymbol{z}_i)) \tag{4}$$

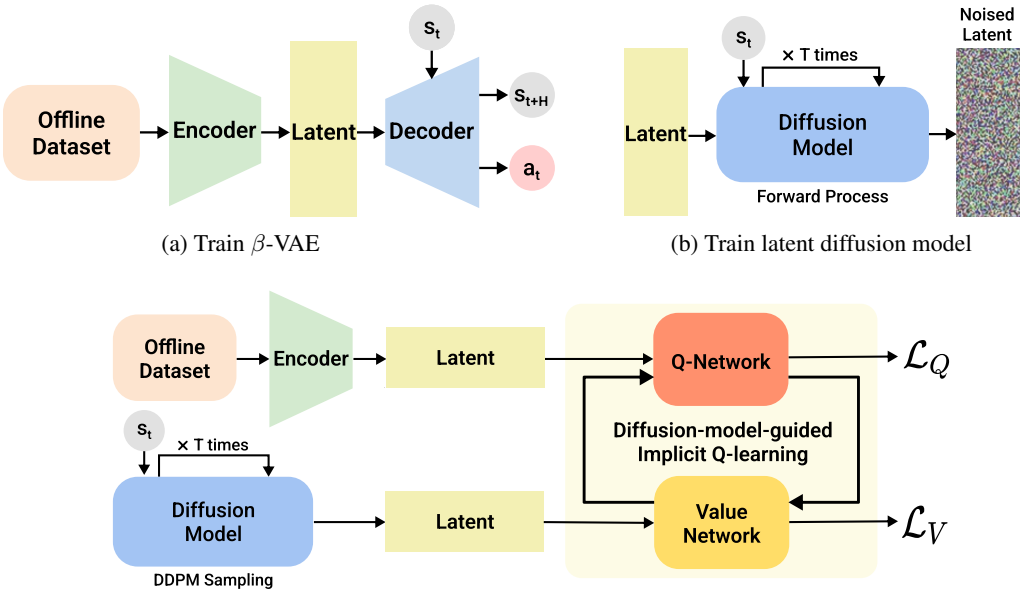
3 Proposed Method

Using diffusion models to address long-horizon tasks typically involves training over the full trajectory length [3]. This approach differs from autoregressive methods that focus on selecting the best action at each step, as it learns the entire action sequence over the horizon. This allows the model to learn long sequences of decisions at once and generate a large number of actions in a single pass. However, predicting decisions across the entire horizon may not always lead to optimal outcomes, as the primary goal is to generate a sequence of decisions corresponding to the sequence length.

Additionally, there is a well-known problem of overestimating the Q-value when training a Q-network [11, 12, 13, 14, 15]. This occurs when certain actions, appearing intermittently, are assigned a high $Q(\boldsymbol{s}, \boldsymbol{a})$ value. In these cases, the state may not actually hold high value, but the Q-value becomes "lucky" and inflated. Therefore, it is essential to ensure that the Q-network does not overestimate and can correctly assess the value based on the current state.

To resolve both of these issues, we propose Diffusion-model-guided Implicit Q-learning with Adaptive Revaluation (DIAR), introducing a value function to assess the value of each situation. Unlike the Q-network, which learns the value of both state and action, the state-value function learns only

the value of the state. By introducing constraints from the value function, we can train a more balanced Q-network and, during the decision-making phase, make more optimal predictions with the help of the value function.



(a) Train β-VAE

(b) Train latent diffusion model

(c) Train Q-network and value-network

Figure 3: Three training stages of DIAR. (a) The β-VAE is trained by encoding a state-action sequence spanning an H-length horizon into a latent space, followed by a policy decoder that outputs actions based on the encoded latent $\boldsymbol{z}$ and the state $\boldsymbol{s}_t$ contained within it. (b) A diffusion model is trained using the encoded latent and the initial state $\boldsymbol{s}_t$. (c) The Q-network is trained on the offline dataset, while the value network is trained on data generated by the diffusion model. This interplay allows the value function and Q-function to guide each other, enabling more balanced learning across both offline samples and generated data.

3.1 Diffusion-model-guided Q-learning Framework

The value-network V_η with parameter η evaluates the value of the current state $\boldsymbol{s}_t$, and the Q-network Q_ϕ with parameter ϕ evaluates the value of the current state $\boldsymbol{s}_t$ and action $\boldsymbol{a}_t$. Additionally, by combining value-network learning with Q-network learning, constraints can be applied to the Q-network, resulting in more balanced training. Instead of relying on the dataset to train the value network and Q-network, we enhance the process by introducing latent vectors generated through a diffusion model. By doing so, we minimize extrapolation errors for unseen decisions in the dataset, leading to more accurate value estimation.

The training of the value-network should aim to reduce the difference between the Q-value and the state-value. Therefore, it is crucial to include the difference between $Q(\boldsymbol{s}, \boldsymbol{z})$ and $V(\boldsymbol{s})$ in the loss function. To achieve this, rather than using MSE loss, we apply weights to make the data distribution more flexible and to respond more sensitively to differences. We use an asymmetric weighted loss function that multiplies the weights of variables u by an expectile factor τ, as shown in Eq. 5. In the next step, u is used as the difference between the Q-value and the state-value for loss calculation.

$$L_\tau^2(u) = |\tau - \mathbb{I}(u < 0)|u^2 \tag{5}$$

By using an asymmetrically weighted loss function, the value-network is trained to reduce the difference between the Q-value and the state-value. We set τ to a value greater than 0.5 and apply Eq. 6 to assign more weight when the difference between the Q-value and the value is large. Additionally,

instead of using latent vector encoded from the dataset, we use latent vectors $\tilde{\boldsymbol{z}}_t$ generated by the diffusion model to guide the learning of a more generalized Q-network.

$$L_V(\eta) = \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{D},\, \tilde{\boldsymbol{z}}_t \sim \mathcal{D}_\psi} \left[L_\tau^2 \left(Q_{\hat{\phi}}(\boldsymbol{s}_t, \tilde{\boldsymbol{z}}_t) - V_\eta(\boldsymbol{s}_t) \right) \right] \tag{6}$$

After the loss for the value-network is calculated, the loss for the Q-network is computed. The loss in Eq. 7 is not based on the Q-network alone but is learned based on the current value and reward, ensuring balance with the value network. The value-network learning, using latent vectors generated by the diffusion model, allows it to handle diverse trajectories, while the Q-network is trained on data pairs $(\boldsymbol{s}_t, \boldsymbol{z}_t, r_{t:t+H}, \boldsymbol{s}_{t+H}) \sim \mathcal{D}$ from the dataset, learning the Q-value of state-latent vector pairs based on existing trajectories. Q-network and value-network training processes form a complementary relationship.

$$L_Q(\phi) = \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{z}_t, r_{t:t+H}, \boldsymbol{s}_{t+H}) \sim \mathcal{D}} \left[\left(r_{t:t+H} + \gamma V_\eta(\boldsymbol{s}_{t+H}) - Q_\phi(\boldsymbol{s}_t, \boldsymbol{z}_t) \right)^2 \right] \tag{7}$$

To ensure stable Deep Q-network training and prevent Q-value overestimation, we employed the Clipped Double Q-learning method [16]. Additionally, we used a prioritized replay buffer $\mathcal{B}$, where the Q-network is trained based on the priority of the samples [17]. $\mathcal{B}$ stores $(\boldsymbol{s}_t, \boldsymbol{z}_t, r_{t:t+H}, \boldsymbol{s}_{t+H})$, which are generated from the offline dataset. The state, action, and reward are taken from the offline dataset, and the latent vector $\boldsymbol{z}_t$ is encoded by $q_{\theta_E}(\boldsymbol{z}|\boldsymbol{s}, \boldsymbol{a})$. The encoded latent vector $\boldsymbol{z}_t$, along with the current state $\boldsymbol{s}_t$, is used to guide the MLP model through the diffusion model to learn the Q-value. The Q-network and value-network are trained alternately, maintaining a complementary relationship through their respective loss functions. The value-network's loss $L_V(\eta)$ is calculated based on the difference between the Q-value and the state-value, which is adjusted by the expectile factor τ. The Q-network's loss $L_Q(\phi)$ is computed using the Bellman equation with the reward and value, where the effect of distant timesteps is controlled by the discount factor γ. The calculated Q-network loss $L_Q(\phi)$ is updated in the model Q_ϕ via backpropagation, and the target Q-network $Q_{\hat{\phi}}$ is gradually updated based on the update rate ρ. The detailed process can be found in Algorithm 1.

Algorithm 1: Diffusion-model-guided Implicit Q-learning with Adaptive Revaluation

Input: Q-network Q_ϕ, target Q-network $Q_{\hat{\phi}}$, value-network V_η, diffusion model $\mu_\psi(\boldsymbol{z}|\boldsymbol{s})$, prioritized replay buffer $\mathcal{B}$, horizon H, number of sampling latent vectors n, latent vector $\boldsymbol{z}$, update rate ρ, max iteration T, learning rate λ_Q, λ_V

$\hat{\phi} \leftarrow \phi$

$t \leftarrow 0$

while $t < T$ **do**

  $(\boldsymbol{s}_t, \boldsymbol{z}_t, r_{t:t+H}, \boldsymbol{s}_{t+H}) \leftarrow \mathcal{B}$

  $\boldsymbol{z}_{t+H}^0, \boldsymbol{z}_{t+H}^1, \ldots, \boldsymbol{z}_{t+H}^{n-1} \leftarrow \mu_\psi(\boldsymbol{z}|\boldsymbol{s}_{t+H})$

  $\eta \leftarrow \eta - \lambda_V \nabla_\eta L_V(\eta)$ # Training value-network

  $\phi \leftarrow \phi - \lambda_Q \nabla_\phi L_Q(\phi)$ # Training Q-network

  $\hat{\phi} \leftarrow \rho\phi + (1-\rho)\hat{\phi}$

  Update priority of $\mathcal{B}$

end

3.2 Adaptive Revaluation in Policy Execution

DIAR method reforms a decision if the value of the current state is higher than the value of the state after making a decision over the horizon length H. We refer to this process as Adaptive Revaluation. Using the value-network V_η, if the current state's value $V(\boldsymbol{s}_t)$ is greater than $V(\boldsymbol{s}_{t+H})$, the value after making a decision for H steps, the method generates a new latent vector $\boldsymbol{z}_t$ from the current

state s_t and continues the decision-making process. When predicting over the horizon length, there may be cases where taking a different action midway through the horizon would be more optimal. In such cases, the value-network V_η checks this, and if the condition is met, a new latent vector is generated.

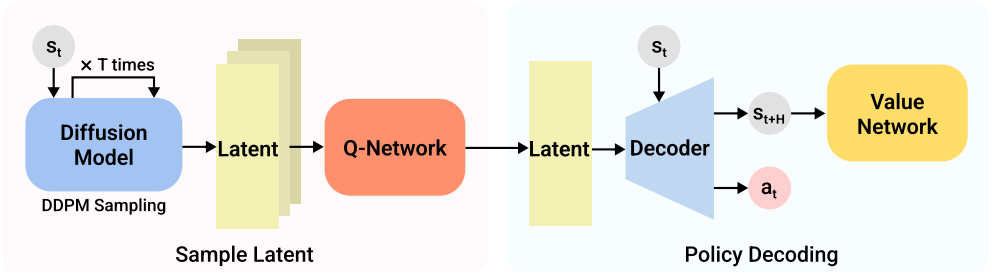
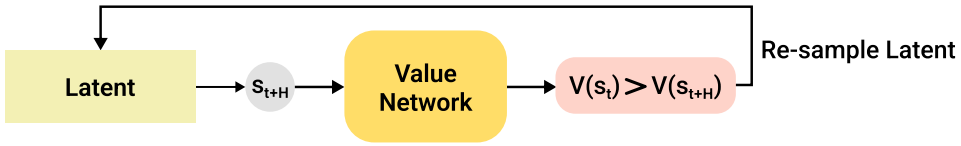
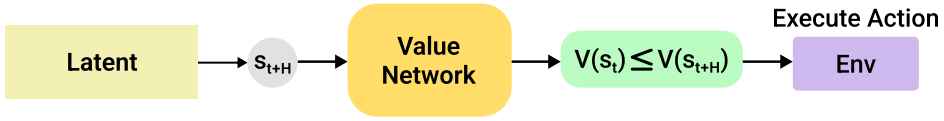


Figure 4: Inference step with DIAR. The current state s_t is put into the diffusion model to extract candidate latent vectors. Then, the latent vector z_t with the highest $Q(s_t, z_t)$ is selected as the best latent vector. This latent vector z_t is subsequently decoded to generate the action a_t. Additionally, the future state s_{t+H} is also decoded to be used for calculating the future value $V(s_{t+H})$.



(a) Non-ideal values for states within a latent in the sparse-reward environment



(b) Ideal values for states within a latent in the sparse-reward environment

Figure 5: The process of finding a better trajectory using Adaptive Revaluation. The process involves making a decision and taking action based on the skill latent z_t with the highest $Q(s_t, z_t)$. The current latent vector z_t is used to predict the future state s_{t+H}, based on which the value $V(s_{t+H})$ of the future state s_{t+H} is calculated. (a) If the value $V(s_t)$ of the current state s_t is greater than the value $V(s_{t+H})$ of the future state s_{t+H}, it is considered non-ideal, and re-sampling is performed. (b) If the value $V(s_{t+H})$ of the future state s_{t+H} is greater than or equal to the value $V(s_t)$ of the current state s_t, it is considered ideal, and the action a_t decoded by the latent vector z_t is executed continuously.

Adaptive Revaluation uses the difference in value to examine whether the agent's predicted decision is optimal. Since the current state s_t can be obtained directly from the environment, it is easy to compute the value $V(s_t)$ of the current state s_t. Whether the current trajectory is optimal can be determined using a state decoder $f_\theta(s_{t+H}|s_t, z_t)$. By inputting the current state s_t and latent vector z_t into the state decoder, the future state s_{t+H} can be predicted. This predicted s_{t+H} is passed into the value-network V_η to estimate its future value $V(s_{t+H})$. By comparing these two values, if the current value $V(s_t)$ is higher, the agent generates new latent vectors and selects the one with the highest $Q(s_t, z_t)$. The detailed Adaptive Revaluation algorithm is shown in Appendix B.

4 Experiments

We compare the performance of our model with other models under various conditions and environments. We focus on goal-based tasks in environments with long-horizons and sparse rewards. For offline RL, we use the Maze2D, AntMaze, and Kitchen datasets to test the strengths of our model in long-horizon sparse reward settings [7]. These environments feature very long trajectories in their

datasets, and rewards are only given upon reaching the goal, making them highly suitable for evaluating our model. We also compare the performance improvements achieved when using Adaptive Revaluation, analyzing whether it allows for reconsideration of decisions when incorrect ones are made and enables the generation of the correct trajectory. Furthermore, to ensure more accurate performance measurements, all scores are averaged over 100 runs and repeated 5 times, with the mean and standard deviation reported.

4.1 Performance on Offline RL Benchmarks

In this section, we compare the performance of our model in offline RL. To evaluate our model, we compare it against various models. These include behavior cloning (BC), which imitates the dataset, and offline RL methods based on Q-learning, such as IQL [18] and IDQL [19]. We also compare our model with DT [20], which uses the transformer architecture employed in LLMs, and methods that use diffusion models, such as Diffuser [3], DD [4], and LDCQ [5]. Through these comparisons with various algorithms, we conduct a quantitative performance evaluation of our model.

Datasets like Maze2D and AntMaze require the agent to learn how to navigate from a random starting point to a random location. Simply mimicking the dataset is insufficient for achieving good performance. The agent must learn what constitutes a good decision and how to make the best judgments throughout the trajectory. Additionally, the ability to stitch together multiple paths through trajectory combinations is essential. In particular, the AntMaze dataset involves a complex state space and requires learning and understanding high-dimensional policies. Our method DIAR consistently demonstrated strong performance in these challenging tasks, where high-dimensional abstraction and reasoning are critical. For more demonstrations, please refer to the Appendix G.

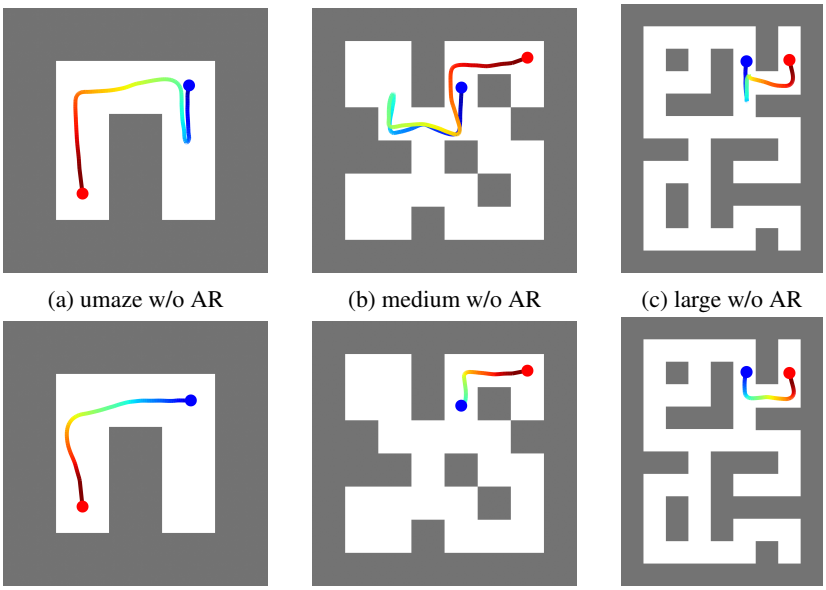
Table 1: Comparison with other methods in long horizon sparse reward D4RL environments.

Dataset	**BC**	**IQL**	**DT**	**IDQL**	**Diffuser**	**DD**	**LDCQ**	**DIAR**
maze2d-umaze-v1	3.8	47.4	27.3	57.9	113.5	-	134.2	141.8±4.3
maze2d-medium-v1	30.3	34.9	32.1	89.5	121.5	-	125.3	139.2±3.5
maze2d-large-v1	5.0	58.6	18.1	90.1	123.0	-	150.1	200.3±3.4
antmaze-umaze-diverse-v2	45.6	62.2	54.0	62.0	-	-	81.4	88.8±1.5
antmaze-medium-diverse-v2	0.0	70.0	0.0	83.5	45.5	24.6	68.9	68.2±6.7
antmaze-large-diverse-v2	0.0	47.5	0.0	56.4	22.0	7.5	57.7	60.6±2.4
kitchen-complete-v0	65.0	62.5	-	-	-	-	62.5	68.8±2.1
kitchen-partial-v0	38.0	46.3	42.0	-	-	57.0	67.8	63.3±0.9
kitchen-mixed-v0	51.5	51.0	50.7	-	-	65.0	62.3	60.8±1.4

4.2 Impact of Adaptive Revaluation

In this section, we analyze the impact of Adaptive Revaluation. We directly compare the cases where Adaptive Revaluation is used and not used in our model. The test is conducted on long-horizon sparse reward tasks, where rewards are sparse. For overall training, an expectile value of $\tau = 0.9$ was used, with $H = 30$ for Maze2D and $H = 20$ for AntMaze and Kitchen. Other training settings were generally the same, and detailed configurations can be found in the Appendix A.

When Adaptive Revaluation is used, it checks whether a better decision might exist according to the value function and discovers a better latent vector to re-create the trajectory. If the value of the current state is higher than the value of a future state, it indicates that a better trajectory might exist than the currently selected decision. This enables the agent to choose a more accurate abstraction and form a more optimal trajectory based on it. The improvement in decision-making with Adaptive Revaluation can be observed in Table 2, which shows how much the agent's decisions improve when using this method.



(a) umaze w/o AR (b) medium w/o AR (c) large w/o AR

(d) umaze w/ AR (e) medium w/ AR (f) large w/ AR

Figure 6: (a)∼(c) Three Maze2D results that only the Q-function is used without Adaptive Revaluation. (d)∼(f) Three Maze2D results for improved decision making using Adaptive Revaluation. Even without Adaptive Revaluation, our model performs well, but we can observe that using Adaptive Revaluation enables more efficient decision-making.

Table 2: Comparison of performance changes with Adaptive Revaluation (AR) in D4RL tasks.

Dataset	**DIAR w/o AR**	**DIAR w/ AR**
maze2d-umaze-v1	135.6±2.8	141.8±4.3
maze2d-medium-v1	138.2±3.1	139.2±3.5
maze2d-large-v1	193.5±4.7	200.3±3.4
antmaze-umaze-diverse-v2	88.8±1.5	85.4±2.6
antmaze-medium-diverse-v2	68.2±6.7	67.4±3.4
antmaze-large-diverse-v2	56.0±4.6	60.6±2.4
kitchen-complete-v0	68.8±2.1	63.8±3.0
kitchen-partial-v0	63.3±0.9	63.0±2.5
kitchen-mixed-v0	60.0±0.7	60.8±1.4

5 Conclusion

In this study, we proposed Diffusion-model-guided Implicit Q-learning with Adaptive Revaluation (DIAR), which leverages diffusion models to enhance abstraction capabilities and train more adaptive agents in offline RL. Our approach consists of two key components. First, we introduced an Adaptive Revaluation algorithm based on the value function, enabling long-horizon predictions while allowing agents to flexibly revise decisions for better outcomes. Second, we developed Diffusion-model-guided Implicit Q-learning to address the challenge of evaluating out-of-distribution state-action pairs in offline RL. By leveraging generative diffusion models, we balance value function and Q-function learning to cover a broader range of scenarios. Through the combination of these methods, DIAR demonstrates strong performance in long-horizon sparse reward tasks including Maze2D, AntMaze, and Kitchen. Notably, DIAR achieves competitive results without requiring extensive task-specific hyperparameter tuning. We believe that latent diffusion models offer significant advantages for offline RL and hold considerable potential for applications across various domains, particularly in robotics.

Acknowledgements

This work was supported by the IITP (RS-2023-00216011), (RS-2024-00445087), (No. 2019-0-01842), (RS-2025-02653113, High-Performance Research AI Computing Infrastructure Support at the 2 PFLOPS Scale) and the NRF (RS-2024-00451162) grants funded by the Ministry of Science and ICT, Korea. GPUs were supported by AICA and GIST SCENT.

References

[1] S. Fujimoto, D. Meger, and D. Precup. Off-Policy Deep Reinforcement Learning without Exploration. In *ICML*, 2019.

[2] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *arXiv:2005.01643*, 2020.

[3] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine. Planning with Diffusion for Flexible Behavior Synthesis. In *ICML*, 2022.

[4] A. Ajay, Y. Du, A. Gupta, J. Tenenbaum, T. Jaakkola, and P. Agrawal. Is Conditional Generative Modeling All You Need for Decision-Making? In *ICLR*, 2023.

[5] S. Venkatraman, S. Khaitan, R. T. Akella, J. Dolan, J. Schneider, and G. Berseth. Reasoning with Latent Diffusion in Offline Reinforcement Learning. In *ICLR*, 2024.

[6] Z. Wang, J. J. Hunt, and M. Zhou. Diffusion Policies as an Expressive Policy Class for Offline Reinforcement Learning. In *ICLR*, 2023.

[7] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4RL: Datasets for Deep Data-Driven Reinforcement Learning. *arXiv:2004.07219*, 2020.

[8] K. Pertsch, Y. Lee, and J. Lim. Accelerating Reinforcement Learning with Learned Skill Priors. In *CoRL*, 2021.

[9] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. In *ICLR*, 2014.

[10] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-Resolution Image Synthesis with Latent Diffusion Models. In *CVPR*, 2022.

[11] H. v. Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-learning. In *AAAI*, 2016.

[12] H. v. Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil. Deep Reinforcement Learning and the Deadly Triad. In *arXiv:1812.02648*, 2018.

[13] J. Fu, A. Kumar, M. Soh, and S. Levine. Diagnosing Bottlenecks in Deep Q-learning Algorithms. In *arXiv:1902.10250*, 2019.

[14] A. Kumar, J. Fu, G. Tucker, and S. Levine. Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. In *NeurIPS*, 2019.

[15] R. Agarwal, D. Schuurmans, and M. Norouzi. An Optimistic Perspective on Offline Reinforcement Learning. In *ICML*, 2020.

[16] S. Fujimoto, H. van Hoof, and D. Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *ICML*, 2018.

[17] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized Experience Replay. In *ICLR*, 2016.

[18] I. Kostrikov, A. Nair, and S. Levine. Offline Reinforcement Learning with Implicit Q-Learning. In *ICLR*, 2022.

[19] P. Hansen-Estruch, I. Kostrikov, M. Janner, J. G. Kuba, and S. Levine. IDQL: Implicit Q-Learning as an Actor-Critic Method with Diffusion Policies. In *arXiv:2304.10573*, 2023.

[20] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision Transformer: Reinforcement Learning via Sequence Modeling. In *NeurIPS*, 2021.

[21] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. In *ICML*, 2015.

[22] J. Ho, A. Jain, and P. Abbeel. Denoising Diffusion Probabilistic Models. In *NeurIPS*, 2020.

A Experiments Details

DIAR consists of three main components: the β-VAE for learning latent skills, the latent diffusion model for learning distributions through latent vectors, and the Q-function, which learns the value of state-latent vector pairs and selects the best latent. These three models are trained sequentially, and when learning the same task, the earlier models can be reused. Detailed model settings and hyperparameters are discussed in the next section. For more detailed code implementation and process, you can refer directly to the code on GitHub.

A.1 β-Variational Autoencoder

The β-VAE consists of an encoder, policy decoder, state prior, and state decoder. The encoder uses two stacked bidirectional GRUs. The output of the GRU is used to compute the mean and standard deviation. Each GRU output is passed through an MLP to calculate the mean and standard deviation, which are then used to compute the latent vector. This latent vector is used by the state prior, state decoder, and policy decoder. The policy decoder takes the latent vector and the current state as input to predict the current action. The state decoder takes the latent vector and the current state to predict the future state. Lastly, the state prior learns the distribution of the latent vector for the current state, ensuring that the latent vector generated by the encoder is trained similarly through KL divergence.

In Maze2D, $H = 30$ is used; in AntMaze and Kitchen, $H = 20$ is used. The diffusion model for the diffusion prior used in β-VAE training employs a transformer architecture. This model differs from the latent diffusion model discussed in the next section, and they are trained independently. Training the β-VAE for too many epochs can lead to overfitting of the latent vector, which can negatively impact the next stage.

Table 3: Hyperparameters for VAE training

Hyperparameter	**Value**
Learning rate	5e-5
Batch size	128
Epochs	100
Latent dimension	16
β	0.1
Diffusion prior steps	200
Optimizer	Adam

A.2 Latent Diffusion Model

The generative model plays the role of learning the distribution of the latent vector for the current state. The current state and latent vector are concatenated and then re-encoded for use. The architecture of the diffusion model follows a U-Net structure, where the dimensionality decreases and then increases, with each block consisting of residual blocks. Unlike the traditional approach of predicting noise ϵ, the diffusion model is trained to directly predict the latent vector z. This process is constrained by Min-SNR-γ. Overall, the diffusion model operates similarly to the DDPM method.

A.3 Q-learning

In our approach, we utilize both a Q-network and a Value network. The Q-network follows the DDQN method, employing two networks that learn slowly according to the update ratio. The Value network uses a single network. Both the Q-network and the Value network are structured with repeated MLP layers. The Q-network encodes the state into a 256-dimensional vector and the latent vector into a 128-dimensional vector. These two vectors are concatenated and passed through additional MLP layers to compute the final Q-value. The Value network only encodes the state into a

Table 4: Hyperparameters for Diffusion model training

Hyperparameter	Value
Learning rate	1e-4
Batch size	128
Epochs	450
Diffusion steps	500
Drop probability	0.1
Min-SNR (γ)	5
Optimizer	Adam

256-dimensional vector, which is then used to compute the value. Between the linear layers, GELU activation functions and LayerNorm are applied. In this way, both the Q-network and Value network are implicitly trained under the guidance of the diffusion model.

Table 5: Hyperparameters for Q-learning

Hyperparameter	Value
Learning rate	5e-4
Batch size	128
Discount factor (γ)	0.995
Target network update rate	0.995
PER buffer α	0.7
PER buffer β	$0.3 \rightarrow 1$
Number of latent samples	500
Expectile (τ)	0.9
extra steps	5
Scheduler	StepLR
Scheduler step	50
Scheduler γ	0.3
Optimizer	Adam

B DIAR Policy Execution Details

We provide a detailed explanation of how DIAR performs policy execution. It primarily selects the latent with the highest Q-value. However, if the current state value $V(\boldsymbol{s}_{t+h})$ is higher than the future state value $V(\boldsymbol{s}_{s+H})$, it triggers another search for a new latent. DIAR repeats this process until it either reaches the goal or the maximum step T is reached.

C Training Process for β-VAE

This section details the process by which the β-VAE is trained. The β-VAE consists of four models: the skill latent encoder, policy decoder, state decoder, and state prior. These four components are trained simultaneously. Additionally, a diffusion prior is trained alongside to guide the β-VAE in generating appropriate latent vectors. The detailed process can be found in Algorithm 3.

D Training Process for Latent Diffusion Model

This section also provides an in-depth explanation of how the latent diffusion model is trained. The goal of the latent diffusion model is to learn the distribution of latent vectors generated by the β-VAE. The latent diffusion model is trained by first converting the offline dataset into latent vectors

Algorithm 2: DIAR Policy Execution

Input: environment *Env*, Q-network $Q(\boldsymbol{s}, \boldsymbol{a})$, value-network $V(\boldsymbol{s})$, policy decoder $\pi_{\theta_D}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z})$, state decoder $f_\theta(\boldsymbol{s}_{t+H}|\boldsymbol{s}_t, \boldsymbol{z}_t)$, diffusion model $\mu_\psi(\boldsymbol{z}|\boldsymbol{s})$, horizon H, max step T, number of sampling latent vectors n, latent vector $\boldsymbol{z}$

$t \leftarrow 0$
$done \leftarrow$ *False*
while not $done$ **do**
  $\boldsymbol{s}_t \leftarrow$ *Env*
  $\boldsymbol{z}_t^0, \boldsymbol{z}_t^1, \ldots, \boldsymbol{z}_t^{n-1} \leftarrow \mu_\psi(\boldsymbol{z}|\boldsymbol{s})$ # Sampling latents vectors from diffusion model
  $Q(\boldsymbol{s}_t, \boldsymbol{z}_t^0), Q(\boldsymbol{s}_t, \boldsymbol{z}_t^1), \ldots, Q(\boldsymbol{s}_t, \boldsymbol{z}_t^{n-1}) \leftarrow Q_\eta(\boldsymbol{s}, \boldsymbol{z})$ # Calculate Q value
  $\boldsymbol{z}_t^i \leftarrow \underset{\boldsymbol{z}_t^i}{\operatorname{argmax}} Q(\boldsymbol{s}_t, \boldsymbol{z}_t^i),\ \boldsymbol{z}^i \in \{\boldsymbol{z}_t^0, \boldsymbol{z}_t^1, \ldots \boldsymbol{z}_t^{n-1}\}$
  $\boldsymbol{s}_{t+H} \leftarrow f_\theta(\boldsymbol{s}_{t+H}|\boldsymbol{s}_t, \boldsymbol{z}_t^i)$ # Predict future state
  $V(\boldsymbol{s}_{t+H}) \leftarrow V_\phi(\boldsymbol{s})$ # Calculate value of future state
  $h \leftarrow 0$
  for $h < H$ **do**
    $\boldsymbol{s}_{t+h} \leftarrow$ *Env*
    $V(\boldsymbol{s}_{t+h}) \leftarrow V_\phi(\boldsymbol{s})$ # Calculate value of current state
    if $V(\boldsymbol{s}_{t+H}) < V(\boldsymbol{s}_{t+h})$ **then**
      break
    end
    else
      $\boldsymbol{a}_{t+h} \leftarrow \pi_{\theta_D}(\boldsymbol{a}_{t+h}|\boldsymbol{s}_{t+h}, \boldsymbol{z}_t^i)$
      Execute action $\boldsymbol{a}_{t+h}$
      Update *done* by *Env*
      $h \leftarrow h + 1$
    end
  end
  $t \leftarrow t + h$
end

using the encoder of the β-VAE, and then learning from these latent vectors. The detailed process can be found in Algorithm 4.

E Theoretical Analysis of DIAR

In this section, we prove that in the case of sparse rewards, when the current timestep t, if the value $V(\boldsymbol{s}_t)$ of the current state $\boldsymbol{s}_t$ is higher than the value $V(\boldsymbol{s}_{t+H})$ of the future state $\boldsymbol{s}_{t+H}$, there is a more ideal trajectory than the current trajectory. An ideal trajectory is defined as one where, for all states at timestep k, the discount factor $0 < \gamma \leq 1$ ensures that $V(\boldsymbol{s}_k) \leq V(\boldsymbol{s}_{k+1})$. This means that for an agent performing actions toward a goal, the value of each state in the trajectory increases monotonically.

Now, consider an assumption about an ideal trajectory: for any timesteps i, j with $i < j$, we assume that $V(\boldsymbol{s}_i) > V(\boldsymbol{s}_j)$ for $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$ from the dataset $\mathcal{D}$. Furthermore, since the state $\boldsymbol{s}_j$ is not the goal and we are in a sparse reward setting, $\forall r(\boldsymbol{s}_i, \boldsymbol{a}_i) = 0$. If we write the Bellman equation for the value function, it results in Eq. 8.

$$V(\boldsymbol{s}_i) = \mathbb{E}_{(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}_{i+1}) \sim \mathcal{D}} \left[r(\boldsymbol{s}_i, \boldsymbol{a}) + \gamma V(\boldsymbol{s}_{i+1}) \right] \tag{8}$$

Algorithm 3: Training Beta Variational Autoencoder

Input: Dataset $\mathcal{D}$, state s_t, action a_t, epoch M, horizon H, diffusion steps T, Min-SNR γ, state prior $p_{\theta_s}(\boldsymbol{z}_t|\boldsymbol{s}_t)$, latent encoder $q_{\theta_E}(\boldsymbol{z}_t|\boldsymbol{s}_{t:t+H}, \boldsymbol{a}_{t:t+H})$, policy decoder $\pi_{\theta_D}(\boldsymbol{a}_{t+i}|\boldsymbol{s}_{t+i}, \boldsymbol{z}_t)$, state decoder $f_\theta(\boldsymbol{s}_{t+H}|\boldsymbol{s}_t, \boldsymbol{z}_t)$, β-VAE parameter θ, diffusion prior μ_ψ, KL regularization coefficient β

$iter \leftarrow 0$

for $iter < M$ **do**

  $\boldsymbol{s}_{t:t+H}, \boldsymbol{a}_{t:t+H} \leftarrow \mathcal{D}$

  $\boldsymbol{z}_t \leftarrow q_{\theta_E}(\boldsymbol{z}_t|\boldsymbol{s}_{t:t+H}, \boldsymbol{a}_{t:t+H})$ # Encoding latent vector

  $\mathcal{L}_1 \leftarrow -\sum_{i=0}^{H-1} \log \pi_{\theta_D}(\boldsymbol{a}_{t+i}|\boldsymbol{s}_{t+i}, \boldsymbol{z}_t)$ # Reconstruction loss

  $\mathcal{L}_2 \leftarrow D_{KL}(q_{\theta_E}(\boldsymbol{z}_t|\boldsymbol{s}_{t:t+H}, \boldsymbol{a}_{t:t+H}) \parallel p_{\theta_s}(\boldsymbol{z}_t|\boldsymbol{s}_t))$ # KL divergence with state prior

  $\mathcal{L}_3 \leftarrow -\log f_\theta(\boldsymbol{s}_{t+H}|\boldsymbol{s}_t, \boldsymbol{z}_t)$ # State decoder loss

  Noise latents $\boldsymbol{z}_j$ from Gaussian noise, $j \sim \mathcal{U}[1, T]$

  $\mathcal{L}_4 \leftarrow \min\{\mathrm{SNR}(j), \gamma\}(\|\boldsymbol{z}_t - \mu_\psi(\boldsymbol{z}_j, \boldsymbol{s}_t, j)\|^2)$ # Diffusion prior loss

  $\mathcal{L}_{total} \leftarrow \mathcal{L}_1 + \beta\mathcal{L}_2 + \mathcal{L}_3 + \mathcal{L}_4$

  Update θ to minimize $\mathcal{L}_{total}$

  $iter \leftarrow iter + 1$

end

Algorithm 4: Training Latent Diffusion Model

Input: Dataset $\mathcal{D}$, state s_t, action a_t, epoch M, horizon H, diffusion steps T, Min-SNR γ, latent encoder $q_{\theta_E}(\boldsymbol{z}|\boldsymbol{s}, \boldsymbol{a})$, diffusion model μ_ψ, variance schedule $\alpha_1, \ldots \alpha_T, \bar{\alpha}_1, \ldots \bar{\alpha}_T, \beta_1, \ldots \beta_T$

$iter \leftarrow 0$

for $iter < M$ **do**

  $\boldsymbol{s}_{t:t+H}, \boldsymbol{a}_{t:t+H} \leftarrow \mathcal{D}$

  $\boldsymbol{z}_t \leftarrow q_{\theta_E}(\boldsymbol{z}_t|\boldsymbol{s}_{t:t+H}, \boldsymbol{a}_{t:t+H})$ # Encoding latent vector

  Sample diffusion time $j \sim \mathcal{U}[1, T]$

  Noise latents from Gaussian noise $\boldsymbol{z}_j \sim \mathcal{N}(\sqrt{\bar{\alpha}_j}\boldsymbol{z}_t, (1 - \bar{\alpha}_j)\mathbf{I})$

  $\mathcal{L} \leftarrow \min\{\mathrm{SNR}(j), \gamma\}(\|\boldsymbol{z}_t - \mu_\psi(\boldsymbol{z}_j, \boldsymbol{s}_t, j)\|^2)$ # Diffusion model loss

  Update ψ to minimize $\mathcal{L}$

  $iter \leftarrow iter + 1$

end

Eq. 8 represents the value function $V(\boldsymbol{s}_i)$ when there is a difference of one timestep. The value function $V(\boldsymbol{s}_i)$ can be computed using the reward received from the action taken in the current state $\boldsymbol{s}_i$ and the value of the next state $\boldsymbol{s}_{i+1}$. Therefore, by iterating Eq. 8 to express the timesteps from i to j, we obtain Eq. 9.

$$V(\boldsymbol{s}_i) = \mathbb{E}_{(\boldsymbol{s}_{i:j}, \boldsymbol{a}_{i:j}) \sim \mathcal{D}} \left[\sum_{t=i}^{j-1} \gamma^{t-i} r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma^{j-i} V(\boldsymbol{s}_j) \right] \tag{9}$$

Since the current environment is sparse in rewards, no reward is given if the goal is not reached. Therefore, in Eq. 9, all reward $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ terms are zero. By substituting the reward as zero and reorganizing Eq. 9, we can derive Eq. 10.

$$V(\boldsymbol{s}_i) = \mathbb{E}_{(\boldsymbol{s}_{i:j}, \boldsymbol{a}_{i:j}) \sim \mathcal{D}} \left[\gamma^{j-i} V(\boldsymbol{s}_j) \right] \tag{10}$$

Since the magnitude of γ is $0 < \gamma \leq 1$, the term $\gamma^{j-i}V(\boldsymbol{s}_j)$ is always less than or equal to $V(\boldsymbol{s}_j)$. This contradicts the initial assumption, indicating that the assumption is incorrect. Therefore, for any ideal trajectory, all value functions $V(\boldsymbol{s}_i)$ must follow a monotonically increasing function. In other words, if the trajectory predicted by the agent is an ideal trajectory, the value $V(\boldsymbol{s}_j)$ after making a decision over the horizon H must always be greater than the current value $V(\boldsymbol{s}_i)$. If the current value $V(\boldsymbol{s}_i)$ is greater than the future value $V(\boldsymbol{s}_j)$, then this trajectory is not an ideal trajectory. Consequently, generating a new latent vector $\boldsymbol{z}_i$ from the current state $\boldsymbol{s}_i$ to search for an optimal decision is a better approach.

F Diffusion Probabilistic Models

Diffusion models [21, 22] function as latent variable generative models, formally expressed through the equation $p_\theta(x_0) := \int p_\theta(x_{0:T}), dx_{1:T}$. Here, $x_1, \ldots, x_T$ denote the sequence of latent variables, integral to the model's capacity to assimilate and recreate the intricate distributions characteristic of high-dimensional data types like images and audio. In these models, the forward process $q(x_t|x_{t-1})$ methodically introduces Gaussian noise into the data, adhering to a predetermined variance schedule delineated by $\beta_1, \ldots, \beta_T$. This step-by-step addition of noise outlines the approximate posterior $q(x_{1:T}|x_0)$ within a structured mathematical formulation, which is specified as follows:

$$\begin{aligned} q(x_{1:T}|x_0) &:= \prod_{t=1}^{T} q(x_t|x_{t-1}), \\ q(x_t|x_{t-1}) &:= \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \end{aligned} \tag{11}$$

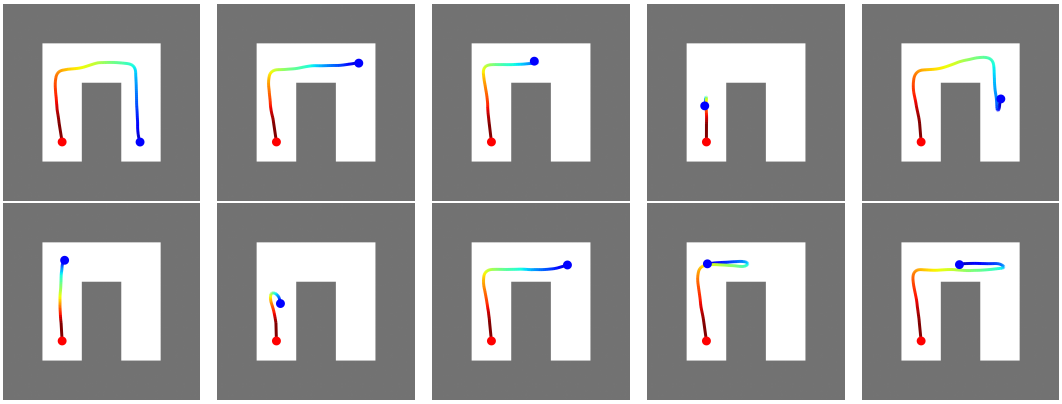
The iterative denoising process, also known as the reverse process, enables sample generation from Gaussian noised data, denoted as $p(x_T) = \mathcal{N}(x_T; 0, I)$. This process is modeled using a Markov chain, where each step involves generating the sample of the subsequent stage from the sample of the previous stage based on conditional probabilities. The joint distribution of the model, $p_\theta(x_{0:T})$, can be represented as follows:

$$\begin{aligned} p_\theta(x_{0:T}) &:= p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t), \\ p_\theta(x_{t-1}|x_t) &:= \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \end{aligned} \tag{12}$$

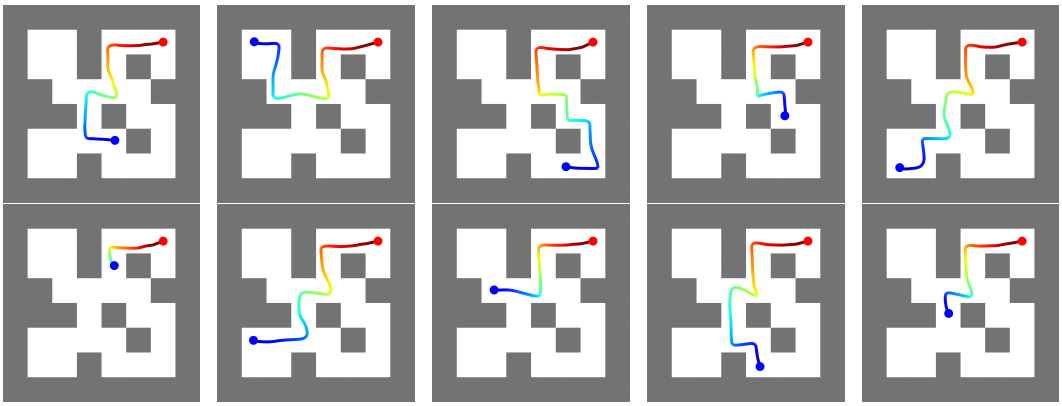
In the Diffusion Probabilistic Model, training is conducted via a reverse process that meticulously reconstructs the original data from noise. This methodological framework allows the Diffusion model to exhibit considerable flexibility and potent performance capabilities. Recent studies have further demonstrated that applying the diffusion process within a latent space created by an autoencoder enhances fidelity and diversity in tasks such as image inpainting and class-conditional image synthesis. This advancement underscores the effectiveness of latent space methodologies in refining the capabilities of diffusion models for complex generative tasks [10]. In light of this, the application of conditions and guidance to the latent space enable diffusion models to function effectively and to exhibit strong generalization capabilities.

G Qualitative Demonstration through Maze2D

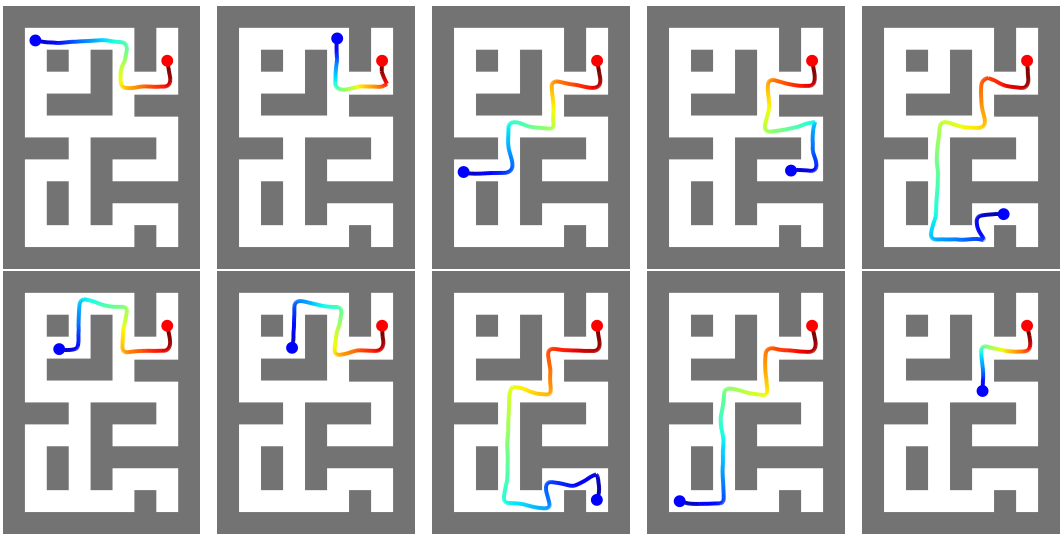
Following the main section, we present additional results in the Maze2D environments, as shown in Figure 7. We qualitatively demonstrate that DIAR consistently generates favorable trajectories.



(a) Maze2D-umaze



(b) Maze2D-medium



(c) Maze2D-large

Figure 7: DIAR-generated trajectories in diverse Maze2D demonstration. DIAR reliably reaches the goal even from starting points (blue) that are far from the goal (red). It even exhibits significant advantages in cases where decisions involve longer horizons.