On the Role of Proposal Support in Diffusion-Based Offline RL for Sequential Decision-Making

Anonymous Author(s)
Affiliation
Address
email

Abstract

Proposal–selection frameworks for offline reinforcement learning (RL) enable decision-making through candidate generation and value-based selection. However, it remains unclear where performance variations arise within this process, particularly across intermediate steps. We study a fundamental limitation of proposal-based decision-making: the extent to which decision quality is constrained by the support of the proposal distribution. Using ARC-AGI as a controlled analysis environment, we construct Synthesized Offline Learning data for Abstraction and Reasoning (SOLAR), a trajectory-based dataset that converts tasks into step-by-step solution trajectories. This enables step-level analysis of decision-making across controlled trajectory distributions. Through experiments with Latent Diffusion-Constrained Q-Learning (LDCQ), we find that decision quality is closely tied to proposal coverage: while the selection mechanism remains reliable when suitable candidates are present, performance degrades sharply when the proposal distribution fails to cover solution-relevant behaviors. These results identify proposal coverage as a key bottleneck in decision-making with diffusion-based offline RL, pointing to directions for improving robustness and generalization.

1 Introduction

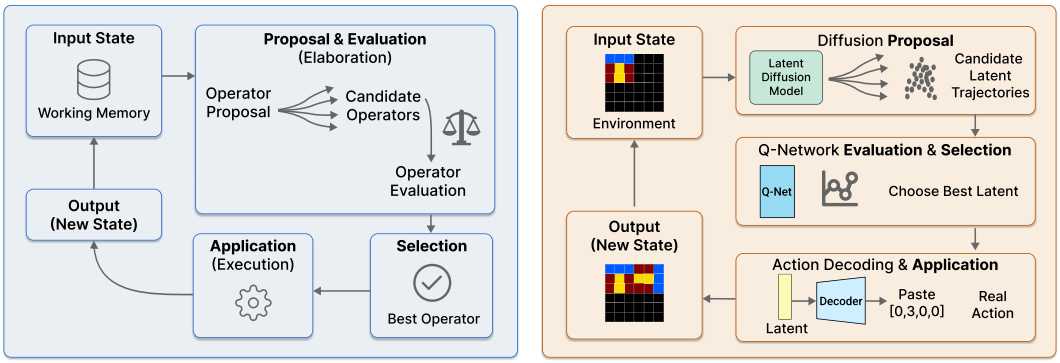
Sequential decision-making in complex environments requires constructing solutions through a series of intermediate steps. Recent approaches to offline reinforcement learning (RL) adopt a proposal–selection structure, where candidate behaviors are generated and evaluated to guide decision-making [2, 14]. In diffusion-based methods, this structure is typically implemented using generative models for proposal and value functions for selection.

It remains unclear what governs decision quality when performance varies across tasks or data conditions. In particular, it is difficult to attribute such variation to the quality of candidate behaviors versus the selection mechanism, as these components are tightly coupled during inference. Disentangling their contributions is essential for diagnosing failure modes and improving the robustness of the proposal–selection-based offline RL.

We study this question using ARC-AGI [1] as a controlled environment in which intermediate state transitions can be explicitly verified, enabling step-level analysis beyond final-outcome evaluation. We construct Synthesized Offline Learning data for Abstraction and Reasoning (SOLAR), a trajectory-based dataset that converts tasks into step-by-step solving trajectories, and evaluate Latent Diffusion-Constrained Q-Learning (LDCQ) [14] within this framework.

Our experiments show that LDCQ constructs multi-step solutions effectively when the trajectory distribution is well-aligned with the task, but performance degrades when the distribution contains sub-optimal trajectories or when the task requires generalization beyond observed ones. We examine this

Submitted to the Workshop on Decision-Making from Offline Datasets to Online Adaptation: Black-Box Optimization to Reinforcement Learning, ICML 2026. Do not distribute.



(a) Cognitive Decision Cycle [8]

(b) Latent Diffusion-Based Offline RL Cycle [14]

Figure 1: A staged decision motif studied in this paper. Both cycles can be decomposed into the phases of proposal, evaluation, selection, and application (execution).

behavior across multiple controlled settings, including single-task expert training, mixed-quality trajectory distributions, and compositional generalization. We find that this degradation stems primarily from insufficient proposal coverage: when the proposal distribution fails to include solution-relevant behaviors, the selection mechanism cannot recover regardless of its accuracy. These findings identify proposal support as the primary bottleneck in diffusion-based offline RL.

Our contributions are as follows:

- We study the role of proposal support in proposal–selection frameworks, isolating the contribution of candidate generation from value-based selection at the step level.
- We introduce SOLAR, a trajectory-based dataset that enables systematic, controlled variation in trajectory quality for fine-grained, step-level analysis of offline RL.
- We provide empirical evidence that proposal coverage is a key bottleneck in diffusion-based offline RL, identifying a concrete target for improving robustness.

2 Background

2.1 Diffusion-based Offline RL

LDCQ [14] models multi-step decision-making using a latent proposal–selection pipeline. This pipeline follows a staged decision cycle consisting of proposal, evaluation, selection, and execution, as illustrated in Figure 1. Trajectory segments $\tau_t = (s_t, a_t, \ldots, s_{t+H-1}, a_{t+H-1})$ are encoded into latent variables z_t, which represent H-step behaviors.

Latent Representation Learning. A β-VAE is trained to encode trajectory segments into latent variables $z_t \sim q_\phi(z_t \mid \tau_t)$ and decode actions via a policy $\pi_\theta(a \mid s, z_t)$. The model is trained to learn compact representations of multi-step behaviors.

Latent Proposal via Diffusion. A conditional diffusion model $p_\psi(z_t \mid s_t)$ is trained over latent variables to generate candidate behaviors given the current state. This mainly serves as the proposal distribution in the decision pipeline.

Value-based Selection. A Q-function $Q(s_t, z_t)$ is trained to evaluate latent behaviors over an H-step horizon. Given a transition from s_t to s_{t+H}, the target value is computed by sampling candidate next latents from the diffusion prior and selecting the one with the highest Q-value:

$$Q(s_t, z_t) \leftarrow r_{t:t+H} + \gamma^H Q\left(s_{t+H}, \operatorname*{argmax}_{z \sim p_\psi(z \mid s_{t+H})} Q(s_{t+H}, z)\right), \tag{1}$$

At inference time, multiple candidates are sampled from $p_\psi(z \mid s_t)$ and the highest-valued one is selected. This enables sequential decision-making through iterative proposal and selection.

2.2 ARC-AGI as a Structured Decision Environment

ARC-AGI [1] is a benchmark where agents must infer transformation rules from a small number of examples and apply them to unseen inputs. Although it is originally formulated as an input–output mapping problem, solving ARC-AGI tasks often involves applying a sequence of transformations to construct the final solution.

We formulate ARC-AGI as a Markov Decision Process, where the state s_t represents the current grid, the operation opr_t specifies the transformation to apply, and the selection sel_t determines where the transformation is applied. The resulting state transition is given by:

$$s_{t+1} = f(s_t, opr_t, sel_t), \tag{2}$$

A reward is provided only upon correct submission, resulting in sparse feedback:

$$r_t = \begin{cases} 1 & \text{if the submitted grid matches the target,} \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

To enable execution and trajectory-level analysis, we use ARCLE [9], a reinforcement learning environment for ARC-AGI that provides explicit state transitions and a discrete action space. ARCLE allows us to simulate sequential decision-making and validate action sequences within an environment.

However, standard ARC-AGI provides only input–output pairs without explicit trajectories, which limits the analysis of intermediate decision processes. This motivates the need for trajectory-level representations to enable step-by-step analysis.

3 SOLAR: A Trajectory-Based Dataset for Analysis

To enable fine-grained analysis of sequential decision-making in ARC-AGI, we introduce the Synthesized Offline Learning data for Abstraction and Reasoning (SOLAR), a trajectory-based dataset that provides explicit state–action transitions for ARC-AGI tasks. SOLAR augments standard ARC-AGI formulations by incorporating executable trajectories, allowing us to observe intermediate decision processes and analyze how solutions are constructed step by step.

This enables systematic investigation of decision-making behavior, including how candidate behaviors are generated, how actions are selected, and how errors emerge across sequential steps. Furthermore, it allows us to analyze not only final outcomes but also the intermediate processes that lead to success or failure. By decoupling task structure from trajectory composition, SOLAR provides a controlled setting for studying how different trajectory distributions affect decision-making behavior.

3.1 Dataset Construction

Each SOLAR task consists of demonstration input–output pairs and a test example, for which an executable episode is provided. The demonstrations define the transformation rule, while the episode specifies step-by-step transitions for solving the test instance.

An episode is represented as $\tau = \{(s_t, a_t, r_t, s_{t+1})\}_{t=0}^{T}$, where s_t is the grid state, a_t is an action, and r_t is a sparse reward provided only upon correct submission. All episodes are executed and validated using ARCLE [9], ensuring that each transition is consistent with a well-defined action space and environment dynamics. For compatibility with diverse types of methods, SOLAR supports both whole-episode and fixed-length segmented formats, accommodating a variety of offline RL methods with temporal abstraction requirements.

3.2 Trajectory Generation

SOLAR is constructed using a rule-based synthesis procedure implemented in SOLAR-Generator. For each task, a solving action sequence is first synthesized to correctly transform the input into the target output, and then validated in ARCLE.

To increase diversity, additional trajectories are generated by varying action orderings or using alternative combinations of operations. The generation process consists of three steps: (i) constructing

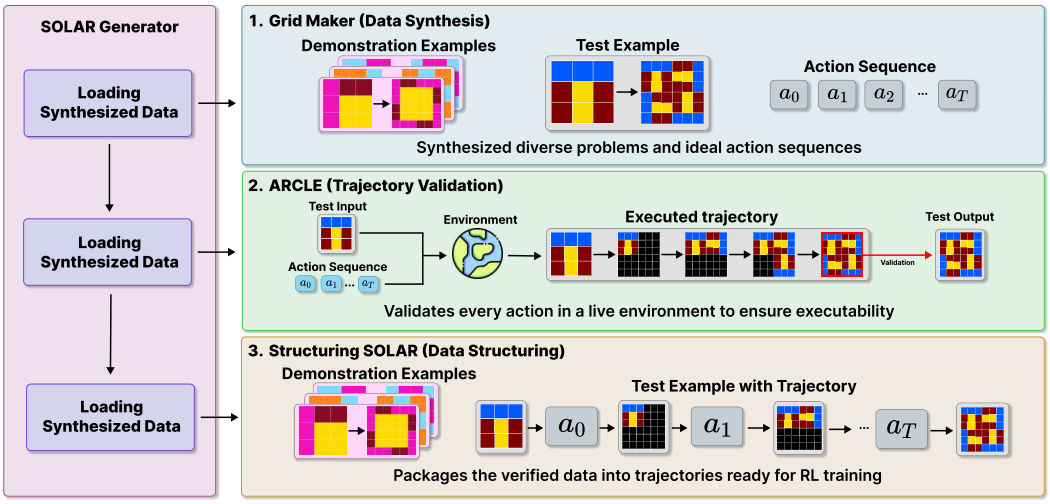


Figure 2: Dataset construction process with SOLAR-Generator. (i) Input-output pairs and action sequences are synthesized, (ii) validated through execution in ARCLE, and (iii) converted into JSON-style episodes for offline RL.

candidate solutions under task constraints, (ii) validating them through execution in ARCLE, and (iii) converting them into transition tuples for offline RL. This results in multiple valid trajectories per task, ranging from direct solutions to longer or suboptimal paths, exposing diverse decision patterns.

Importantly, SOLAR supports flexible construction of trajectory distributions by combining solution-oriented and suboptimal trajectories. This allows controlled analysis of how decision-making behavior changes under different trajectory conditions while keeping the underlying task fixed. Additional implementation details are provided in Appendix B.

4 Experiments

4.1 Experimental Setup

All experiments are conducted using the SOLAR dataset, which provides executable state–action trajectories for ARC-AGI tasks. The agent interacts with the environment by applying actions sequentially, receiving a reward only upon correct submission.

To control the trajectory distribution, we construct offline datasets by mixing expert trajectories $\mathcal{D}_{\text{exp}}$ and suboptimal trajectories $\mathcal{D}_{\text{sub}}$:

$$\mathcal{D}(\alpha) = \alpha\mathcal{D}_{\text{exp}} + (1-\alpha)\mathcal{D}_{\text{sub}} \tag{4}$$

where $\alpha \in [0, 1]$ controls the proportion of expert trajectories. Suboptimal trajectories may either reach the correct solution via a longer path or fail to reach it entirely. We consider $\alpha = 1$ (expert-only) and $\alpha = 1/2$ (mixed-quality) settings.

For each task, we synthesize 5,000 training episodes and 100 test episodes using SOLAR-Generator, with non-overlapping test examples. We report task success rate, defined as the proportion of episodes that terminate with a correct final output.

4.2 Behavior under Expert Trajectories

We evaluate LDCQ in a controlled, single-task setting in which the trajectory distribution is fully aligned with the task, using expert-only trajectories. In this setting, all training episodes successfully reach the correct solution, allowing us to isolate the agent's ability to learn valid multi-step transformations.

Table 1 summarizes performance across representative ARC-AGI tasks. While LDCQ achieves near-perfect accuracy on several tasks, performance varies depending on task structure. Notably, some tasks such as `0d3d703e` exhibit near-zero performance despite expert-only training.

These tasks require inferring global transformation rules (e.g., color permutations) that are not explicitly reflected in intermediate state transitions, making them difficult to capture through step-level trajectory representations. In contrast, tasks involving spatial transformations (e.g., coloring specific regions) yield more direct state changes that are easier to model.

Overall, LDCQ performs reliably when solutions can be decomposed into explicit sequences of intermediate transformations, confirming that the model can learn effective multi-step behaviors when the trajectory distribution provides sufficient coverage.

Table 1: Single-task performance with expert-only trajectories.

Task ID	Accuracy	Task ID	Accuracy
5c0a986e-colorfix	100%	a65b410d	100%
4c4377d9	98%	6f8cd79b	91%
4258a5f9	79%	007bbfb7	77%
46442a0e	76%	5c0a986e-colordiff	68%
74dd1130	24%	0d3d703e	0%

4.3 Selecting Efficient Solution Paths

We examine whether LDCQ can identify more efficient behaviors when multiple solutions exist. To study this, we construct a controlled SOLAR dataset for task `a65b410d`, where all trajectories reach the correct solution but differ in efficiency. As illustrated in Figure 3, the two training trajectories diverge before a shared intermediate state and reconverge after it: one takes an optimal path in the first half but an inefficient path in the second, and the other does the reverse. As a result, no single training trajectory is fully optimal end-to-end, and a fully efficient solution can only be constructed by combining efficient segments from different trajectories.

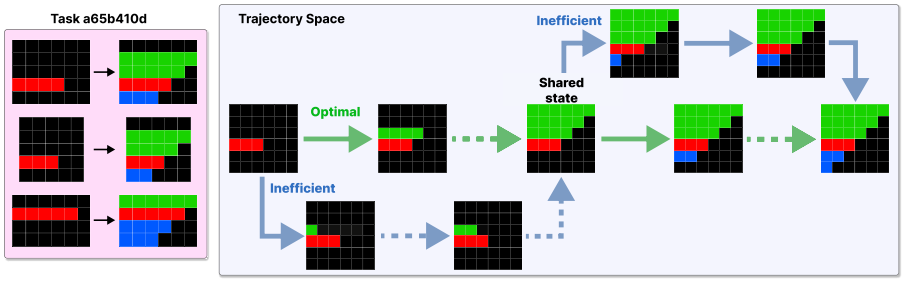


Figure 3: Two ways of solving Task `a65b410d`. The two trajectories share an intermediate state, with each being optimal in one half and inefficient in the other. A fully efficient solution requires stitching the optimal segments across trajectories, which no single training episode achieves end-to-end.

We compare LDCQ against two ablated baselines that omit selection: VAE Prior and Diffusion Prior, which use a single candidate sampled from the VAE and from the diffusion model, respectively. As shown in Table 2, all methods achieve high accuracy, but LDCQ achieves with fewer steps.

Table 2: Comparison of policies on a task where all trajectories reach correct solutions but differ in efficiency. LDCQ significantly reduces the number of steps, highlighting the role of selection.

Policy	**Accuracy**	**Avg. Steps**
VAE Prior	90%	19.4
Diffusion Prior	99%	15.4
LDCQ (w/ Q)	100%	**7.0**

While generative policies (VAE prior, Diffusion prior) tend to follow training trajectories and produce longer solutions, LDCQ achieves the same accuracy with significantly fewer steps. This suggests that value-based selection enables the model to prefer more efficient action sequences rather than simply reproducing observed trajectories.

4.4 Behavior under Mixed-quality Trajectories

We next consider a multi-task setting where the dataset contains a mixture of expert and suboptimal trajectories. We vary α to control the proportion of expert trajectories and evaluate how performance changes across methods.

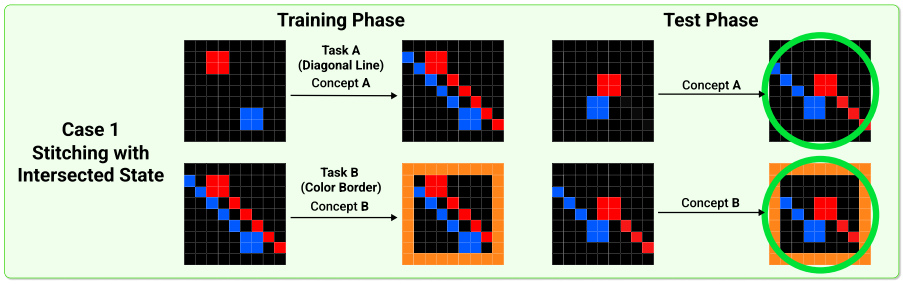
Table 3: Baseline comparison on multi-task learning under expert-only ($\alpha = 1$) and mixed-quality ($\alpha = 1/2$) training.

Method	Expert-only	Mixed-quality
BC	38.8±1.86%	9.67±2.44%
CQL [7]	61.9±0.31%	**37.5±0.31%**
IQL [6]	58.3±0.31%	32.4±1.11%
LDCQ [14]	**66.9±6.39%**	23.7±6.04%

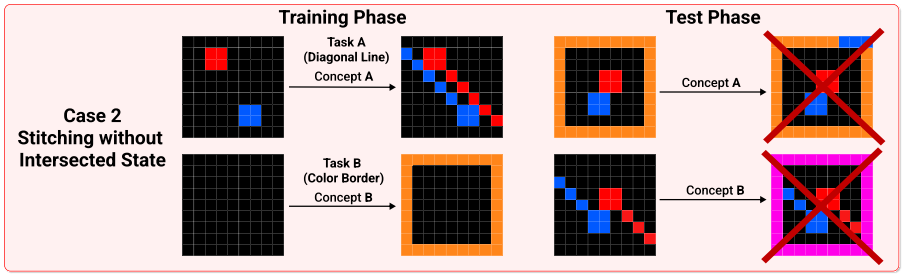
As shown in Table 3, LDCQ achieves the highest performance under expert-only data, but its performance drop under mixed-quality trajectories is notably larger than that of other methods. This higher sensitivity to distributional variation may be attributed to the nature of latent-based proposal generation: unlike CQL and IQL, which apply explicit action-level constraints or regularization to stabilize learning, LDCQ relies on a diffusion prior over latent trajectory segments, making it more susceptible to degradation when the training distribution contains suboptimal behaviors. Qualitatively, the agent often reaches states close to the target but fails to complete the task, producing inconsistent action sequences.

These results suggest that decision quality is closely tied to the availability of suitable candidate behaviors in the proposal stage, which we analyze in detail in Section 5.

4.5 Compositional Generalization



(a) Applying learned concepts to seen intermediate states (Case 1)



(b) Applying learned concepts to unseen intermediate states (Case 2)

Figure 4: Compositional generalization with and without intermediate state support. (a) When the intermediate state at the composition boundary is covered by training data, the agent successfully composes the two learned transformations. (b) When it is absent, composition fails despite each transformation having been learned individually, highlighting that generalization is constrained by trajectory-level support rather than concept-level learning.

We examine whether LDCQ can generalize to compositional tasks that require combining independently learned transformations. The agent is trained separately on two individual transformations,

Table 4: Effect of trajectory support on compositional generalization.

Condition	State Seen	Transition Seen	Accuracy
Seen task	Yes	Yes	100%
Unseen state	No	Partial	0–33%
Composition (with shared states)	Yes	Yes	100%
Composition (without shared states)	No	No	0%

Task A (diagonal line) and Task B (color border), and evaluated on a compositional task that requires executing them sequentially. We consider two cases depending on whether the training episodes share an intermediate state at the composition boundary: in Case 1, the intermediate state after applying Task A appears in the training data for Task B, while in Case 2 it does not. As a baseline, we also evaluate each transformation individually on unseen intermediate states (Case 2 without composition), finding that performance already degrades (0–33%) even before attempting composition, confirming that the agent struggles to generalize to states outside the training distribution.

As shown in Figure 4 and Table 4, composition succeeds in Case 1 where the intermediate state is covered by training data, but fails in Case 2 where it is not, even though each transformation has been learned individually. When transitions are only partially supported, performance varies across concepts, with Concept A failing completely (0%) and Concept B achieving limited success (33%).

These results indicate that compositional generalization is fundamentally constrained by the availability of compatible trajectory segments in the proposal distribution: even when individual transformations are learned, the agent cannot compose them if the necessary intermediate states and transitions are absent from the training data.

5 Analysis

In this section, we analyze the failure modes observed across our experiments to identify the underlying factors that govern the success and limitations of proposal-based decision-making.

5.1 Failure Decomposition

As observed in Section 4.4, LDCQ exhibits significant performance degradation under mixed-quality data. To better understand this behavior, we analyze how failures occur during the decision process. We observe that the agent often reaches states that are close to the target but fails to consistently complete the task. In many cases, early-stage transitions are reasonable, but errors accumulate in later steps, leading to incorrect final outputs.

Moreover, generated episodes frequently contain locally valid transitions but fail to form coherent sequences toward the goal. This suggests that failures are not caused by isolated incorrect actions, but by inconsistencies across sequential decisions. These observations motivate a separate analysis of the proposal and selection stages, which we examine next.

5.2 Diagnosing the Source of Failure

To identify the source of failure, we analyze the proposal and selection stages separately. LDCQ follows a proposal–selection pipeline, where candidate behaviors are generated from a diffusion prior and evaluated using a learned Q-function.

We first examine whether the Q-network fails to correctly select among candidate behaviors. To isolate this effect, we perform an oracle proposal analysis, where the correct action is explicitly included in the candidate set. Since SOLAR provides ground-truth trajectories for test instances, we encode the corresponding trajectory segments using the trained β-VAE to obtain the latent representation of the correct behavior, and inject this latent into the candidate set during inference. For each of 500 randomly sampled test states, we generate 100 latent candidates from the diffusion prior, check whether the oracle latent is included, and evaluate whether the Q-network selects it when present.

Table 5: Proposal coverage and conditional selection accuracy.

	Expert-only	Mixed-quality
$P(\text{oracle} \in \text{proposals})$	91.0%	62.2%
$P(\text{correct action} \mid \text{oracle})$	95.6%	88.4%

As shown in Table 5, the probability that the correct candidate is included in the proposal set drops notably under mixed-quality data (91.0% $\rightarrow$ 62.2%). However, when the correct candidate is available, the Q-network selects it with high accuracy (88.4%), even under mixed-quality data.

These results indicate that the selection mechanism remains reliable when suitable candidates are present. However, this reliability is conditioned on the quality of the proposal distribution, and the primary source of failure is the limited coverage of the proposal distribution rather than inaccuracies in value-based selection.

5.3 Proposal Support Limitation

We analyze how often correct candidates appear in the proposal set across states. As shown in Table 6, under mixed-quality data, a large proportion of states (37.8%) contain no correct candidates at all. This indicates that the correct behavior is entirely absent from the proposal set, forcing the agent to select among suboptimal options regardless of the accuracy of the Q-function.

Table 6: Distribution of oracle candidates across states (100 proposals), showing that many states lack any correct candidate under mixed-quality data.

# Oracle proposals	Expert-only	Mixed-quality
0	9.0%	**37.8%**
1–20	1.4%	37.2%
20–50	2.4%	8.4%
50–99	10.8%	6.8%
100	**76.4%**	9.8%

Increasing the number of proposals improves the chance of including correct candidates, but also introduces more out-of-distribution behaviors. As shown in Table 7, performance improves as the number of proposals increases, but degrades beyond a certain point. These out-of-distribution candidates are not well calibrated by the Q-network, causing selection to become biased toward unreliable candidates and leading to degraded performance despite increased proposal diversity.

Table 7: Effect of the number of proposals on performance ($\alpha = 1/2$).

# Proposals	Accuracy (%)
1	5.6
10	18.2
50	**26.2**
100	25.4
500	20.0
1000	19.8

This indicates that proposal limitations arise not only from insufficient coverage, but also from the quality of the proposal distribution. When correct candidates are absent, the agent is forced to select among suboptimal options; when too many out-of-distribution candidates are present, selection itself becomes less reliable.

Together, these results reveal a structural limitation of the LDCQ framework: decision quality is fundamentally constrained by the support and quality of the proposal distribution. Even a well-trained value function cannot recover correct behavior when suitable candidates are absent or poorly represented.

6 Discussion: Towards Better Proposal Support

Our analysis reveals that decision quality in diffusion-based offline RL is fundamentally constrained by the support of the proposal distribution. This manifests in three failure modes: insufficient coverage under mixed-quality data, degraded selection under out-of-distribution proposals, and failure of compositional generalization. We discuss each and outline directions for addressing these limitations.

Improving proposal coverage under data limitations. A substantial fraction of states lack correct candidates under mixed-quality data, as the diffusion prior $p_\psi(z_t \mid s_t)$ is trained to replicate the offline dataset, which may itself be incomplete or noisy. This suggests that proposal diversity is fundamentally constrained by data support. One direction is to decouple proposal generation from raw data quality, for example, through latent-space augmentation, retrieval-augmented proposals, or filtering of suboptimal trajectories. More broadly, this highlights the need for mechanisms that can enrich the proposal distribution beyond the empirical dataset.

Calibrating selection and decoding under unreliable proposals. Increasing the number of proposals introduces out-of-distribution latents that are not reliably evaluated by the Q-network, leading to overestimation bias. This reflects a fundamental tension between proposal coverage and selection reliability. More broadly, the current formulation tightly couples latent representations with action decoding, meaning that even directionally correct latents may fail to produce valid action sequences during execution. This suggests that latent representations are better interpreted as high-level directional guidance rather than precise action specifications, and should be coupled with more expressive decoding mechanisms that ground them into executable actions conditioned on the current state.

Compositional generalization and latent structure. Our results show that composition fails when intermediate states or transitions are not covered by the training data. This limitation is closely tied to the current trajectory-level latent representation, which encodes fixed segments of behavior rather than reusable primitives. Structuring the latent space to capture modular or compositional behaviors, rather than monolithic trajectory segments, may enable more flexible recombination and improve generalization beyond the training distribution.

Towards hybrid offline-to-online adaptation. The limitations identified above reflect a fundamental constraint of purely offline learning: the proposal distribution is inherently bounded by the support of the dataset, and correct behaviors cannot be recovered when they are absent from the offline data, regardless of the quality of the value function. Our results show that the model performs well on seen distributions but struggles to generalize, highlighting the need for mechanisms that actively expand proposal support beyond what offline data provides. Limited online interaction offers a principled way to address this gap, by exploring regions where offline coverage is sparse and augmenting the proposal distribution with behaviors that cannot be inferred from static datasets alone.

7 Conclusion

We studied diffusion-based offline RL with a focus on proposal–selection decision-making, using ARC-AGI as a controlled analysis environment. Using the SOLAR dataset, we conducted a systematic analysis across single-task, mixed-quality, and compositional settings. Our results show that LDCQ constructs multi-step solutions effectively when the trajectory distribution is well-aligned with the task, but performance degrades when the distribution contains suboptimal trajectories or when the task requires generalization beyond observed ones. Through detailed analysis, we find that this degradation is not primarily due to inaccuracies in value-based selection, but is closely tied to the availability of candidate behaviors, identifying proposal coverage as a key bottleneck in proposal–selection-based offline RL. The SOLAR dataset and diagnostic methodology introduced in this work provide a controlled testbed for future approaches targeting improved proposal coverage, uncertainty-aware selection, and compositional generalization.

References

[1] François Chollet. On the Measure of Intelligence. *arXiv:1911.01547*, 2019.

[2] Scott Fujimoto, David Meger, and Doina Precup. Off-Policy Deep Reinforcement Learning without Exploration. In *ICML*, 2019.

[3] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *ICML*, 2018.

[4] Tiankai Hang, Shuyang Gu, Chen Li, Jianmin Bao, Dong Chen, Han Hu, Xin Geng, and Baining Guo. Efficient Diffusion Training via Min-SNR Weighting Strategy. In *ICCV*, 2023.

[5] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *NeurIPS*, 2020.

[6] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline Reinforcement Learning with Implicit Q-Learning. In *ICLR*, 2022.

[7] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-Learning for Offline Reinforcement Learning. In *NeurIPS*, 2020.

[8] John E Laird, Allen Newell, and Paul S Rosenbloom. Soar: An Architecture for General Intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[9] Hosung Lee, Sejin Kim, Seungpil Lee, Sanha Hwang, Jihwan Lee, Byung-Jun Lee, and Sundong Kim. ARCLE: The Abstraction and Reasoning Corpus Learning Environment for Reinforcement Learning. In *CoLLAs*, 2024.

[10] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical Text-Conditional Image Generation with CLIP Latents. *arXiv preprint arXiv:2204.06125*, 2022.

[11] Tom Schaul. Prioritized Experience Replay. In *ICLR*, 2016.

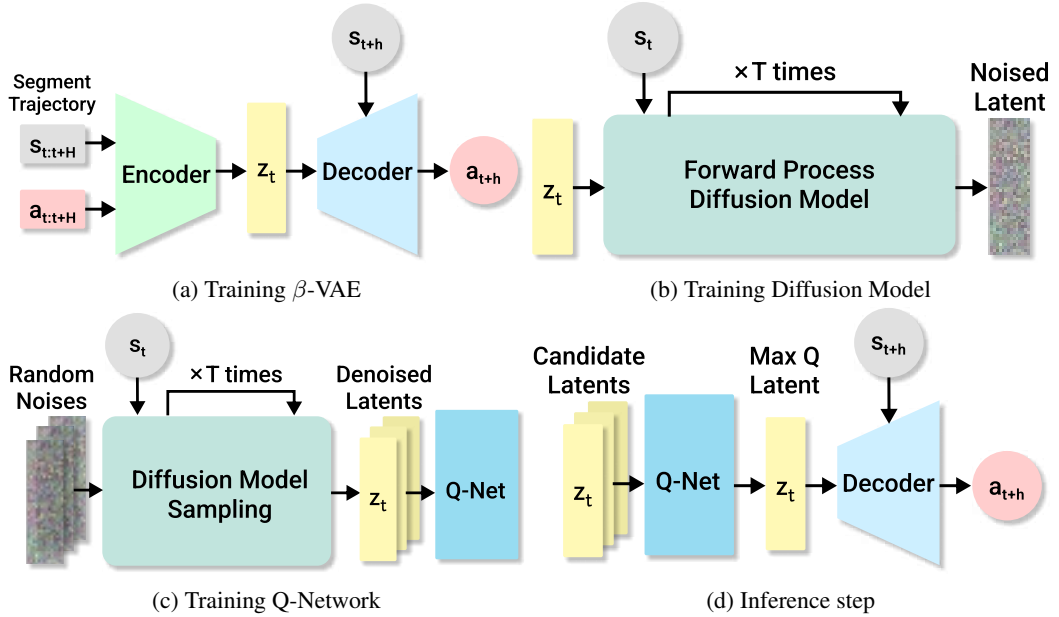
[12] Suyeon Shim, Dohyun Ko, Hosung Lee, Seokki Lee, Doyoon Song, Sanha Hwang, Sejin Kim, and Sundong Kim. O2ARC 3.0: A Platform for Solving and Creating ARC Tasks. In *IJCAI*, 2024.

[13] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising Diffusion Implicit Models. 2020.

[14] Siddarth Venkatraman, Shivesh Khaitan, Ravi Tej Akella, John Dolan, Jeff Schneider, and Glen Berseth. Reasoning with Latent Diffusion in Offline Reinforcement Learning. In *ICLR*, 2024.

A Training Details

Training Latent Encoder and Policy Decoder The first stage in training with LDCQ is to train a β-VAE that learns latent representations. In this stage, the β-VAE learns how actions are executed over multiple steps to change the state. With H-horizon latents, it becomes easier to capture longer-term changes in the state. We use SOLAR as the training dataset $\mathcal{D}$, which contains H-length segmented trajectories $\boldsymbol{\tau}_t$. Each $\boldsymbol{\tau}_t$ consists of state sequences $\boldsymbol{s}_{t:t+H} = [\boldsymbol{s}_t, \cdots, \boldsymbol{s}_{t+H-1}]$ and action sequences $\boldsymbol{a}_{t:t+H} = [\boldsymbol{a}_t, \cdots, \boldsymbol{a}_{t+H-1}]$, along with additional information such as demonstration examples. As shown in Figure 5a, during the β-VAE training stage, the encoder q_ϕ is trained to encode $\boldsymbol{\tau}_t$ into the latent representation $\boldsymbol{z}_t$, and the low-level policy decoder π_θ is trained to decode actions based on the given state and latent. For example, given the latent $\boldsymbol{z}_t$ and a state from the segment trajectory, $\boldsymbol{s}_{t+h}$ where $h \in [0, H)$, the policy decoder decodes the action $\boldsymbol{a}_{t+h}$ for $\boldsymbol{s}_{t+h}$. The β-VAE is trained by maximizing the evidence lower bound (ELBO), minimizing the loss in Eq. 5. The loss consists of the reconstruction loss from the low-level policy decoder and the KL divergence between the approximate posterior $q_\phi(\boldsymbol{z}_t|\boldsymbol{\tau}_t)$ and the prior $p_\omega(\boldsymbol{z}_t|\boldsymbol{s}_t)$.



(a) Training β-VAE (b) Training Diffusion Model

(c) Training Q-Network (d) Inference step

Figure 5: (a)–(c) Training stages of LDCQ. (a) Training a β-VAE with an encoder that encodes H-horizon segment trajectories into latents $\boldsymbol{z}_t$, and a policy decoder that decodes actions based on $\boldsymbol{z}_t$ and state $\boldsymbol{s}_{t+h}$ where $h \in [0, H)$ contained in the latent. (b) Training a diffusion model based on $\boldsymbol{z}_t$ and the $\boldsymbol{s}_t$. (c) Training a Q-network using latents sampled through the diffusion model. (d) LDCQ inference step at $\boldsymbol{s}_{t+h}$. Possible latents at $\boldsymbol{s}_t$ are sampled through the diffusion model, and the agent executes actions resulting from decoding the latent with the highest Q-value.

$$\mathcal{L}_{\text{VAE}}(\theta, \phi, \omega) = -\mathbb{E}_{\boldsymbol{\tau}_t \sim \mathcal{D}} \Bigg[\mathbb{E}_{q_\phi(\boldsymbol{z}_t|\boldsymbol{\tau}_t)} \Bigg[\sum_{l=t}^{t+H-1} \log \pi_\theta(\boldsymbol{a}_l|\boldsymbol{s}_l, \boldsymbol{z}_t) \Bigg] - \beta \cdot D_{KL}\Big(q_\phi(\boldsymbol{z}_t|\boldsymbol{\tau}_t) \parallel p_\omega(\boldsymbol{z}_t|\boldsymbol{s}_t)\Big) \Bigg] \tag{5}$$

Training Latent Diffusion Model In the second stage, latent diffusion model is trained to generate latents based on the latent representations encoded by the β-VAE. The training data consists of $(\boldsymbol{s}_t, \boldsymbol{z}_t)$ pairs, which are used to train a conditional latent diffusion model $p_\psi(\boldsymbol{z}_t|\boldsymbol{s}_t)$ by learning the denoising function $\mu_\psi(\boldsymbol{z}_t^j, \boldsymbol{s}_t, j)$, where $j \in [0, T]$ is diffusion timestep. This allows the model to capture the distribution of trajectory latents conditioned on $\boldsymbol{s}_t$. $q(\boldsymbol{z}_t^j|\boldsymbol{z}_t^0)$ denotes the forward Gaussian diffusion process that noising the original data. Following previous research [10, 14], we predict the original latent rather than the noise, balancing the loss across diffusion timesteps using the Min-SNR-γ strategy [4]. The loss function used to train the diffusion model is shown in Eq. 6. Here, $\boldsymbol{z}_t^j$, $j \in [0, T]$ represents noised latent on j-th diffusion time step, when $j = 0$ then $\boldsymbol{z}_t^0 = \boldsymbol{z}_t$ and $\boldsymbol{z}_t^T$ is Gaussian noise.

$$\mathcal{L}(\psi) = \mathbb{E}_{\substack{j \sim [1,T],\, \boldsymbol{\tau}_H \sim \mathcal{D}, \\ \boldsymbol{z}_t \sim q_\phi(\boldsymbol{z}_t|\boldsymbol{\tau}_t), \\ \boldsymbol{z}_t^j \sim q(\boldsymbol{z}_t^j|\boldsymbol{z}_t^0)}} \Big[\min\{\text{SNR}(j), \gamma\} \cdot \|\boldsymbol{z}_t^0 - \mu_\psi(\boldsymbol{z}_t^j, \boldsymbol{s}_t, j)\|^2 \Big] \tag{6}$$

Training Q-Network Finally, the latent vectors sampled by the latent diffusion model are used for Q-learning. For latent sampling, while the original LDCQ framework uses DDPM sampling [5], we instead adopt DDIM sampling [13] to accelerate latent generation with fewer denoising steps. The trained diffusion model samples latents by denoising random noise conditioned on the state information $\boldsymbol{s}_t$. We use the data consisting of $(\boldsymbol{s}_t, \boldsymbol{z}_t, r_{t:t+H}, \boldsymbol{s}_{t+H})$ for training Q-network, where

$r_{t:t+H} = \sum_{l=t}^{t+H-1} \gamma^l r_l$ denotes the discounted sum of rewards. Here, DDIM sampling is used to sample $\boldsymbol{z}_{t+H}$ conditioned on $\boldsymbol{s}_{t+H}$. For Q-learning, we use Clipped Double Q-learning [3] as shown in Eq. 7 with Prioritized Experience Replay buffer [11] to improve learning stability and mitigate overestimation. The trained Q-network $Q(\boldsymbol{s}_t, \boldsymbol{z}_t)$ evaluates the expected return of performing various H-length actions, with $\boldsymbol{z}_t$ sampled based on $\boldsymbol{s}_t$. This allows the network to efficiently calculate the value of actions over H-steps to estimate future returns. Additionally, since ARC-AGI tasks involve inferring analogies from demonstration pairs, the embedded representation of the demonstration pair, $\boldsymbol{p}_{emb}$, is also used in the Q-function calculation.

$$
\begin{aligned}
&Q(\boldsymbol{s}_t, \boldsymbol{z}_t, \boldsymbol{p}_{emb}) \\
&\leftarrow \left(r_{t:t+H} + \gamma^H Q\Big(\boldsymbol{s}_{t+H}, \underset{\boldsymbol{z} \sim p_\psi(\boldsymbol{z}_{t+H} | \boldsymbol{s}_{t+H})}{\operatorname{argmax}} Q(\boldsymbol{s}_{t+H}, \boldsymbol{z}, \boldsymbol{p}_{emb}), \boldsymbol{p}_{emb}\Big) \right)
\end{aligned} \tag{7}
$$

A.1 Hyperparameters

We used a horizon length of 5 for encoding skill latents, meaning the model plans and evaluates actions over a five-step lookahead.

We trained the diffusion model with 500 diffusion steps. If the number of diffusion steps is too small, it can lead to high variance in the sampling process, potentially causing errors during the decoding of operations or selections in ARCLE. To minimize these errors, we set the number of diffusion steps to 500, ensuring more accurate operation and selection decoding from the sampled latents.

We set the discount factor to 0.5 to ensure the model appropriately balances immediate and future rewards. Since the total steps required to reach the correct answer in ARCLE are usually fewer than 20, a high discount factor could cause the agent to struggle in distinguishing between submitting at the correct state and continuing with additional steps, which could lead to episode failure.

The hyperparameters that we used for training three stages of LDCQ are shown in Tables 8, 9 and 10.

Table 8: Hyperparameters for training β-VAE

Parameter	**Value**
Learning rate	5e-5
Batch size	128
Epochs	400
Horizon (H)	5
Latent dimension (z)	256
KL loss ratio (β)	0.1
Hidden layer dimension	512

Table 9: Hyperparameters for training latent diffusion model

Parameter	**Value**
Learning rate	1e-4
Batch size	64
Epochs	400
Diffusion steps (T)	500
Variance schedule	linear
Sampling algorithm	DDIM
γ (For Min-SNR-γ weighting)	5

A.2 Hardware

We used an NVIDIA A100-SXM4-40GB GPU to train the model. Training the β-VAE took about 7 hours, while training the diffusion model and Q-network each took around 6 to 10 hours.

Table 10: Hyperparameters for training DQN

Parameter	Value
Learning rate	5e-4
Batch size	64
Discount factor (γ)	0.5
Target net update rate (ρ)	0.995
PER buffer α	0.7
PER buffer β range	$[0.3, 1)$
PER buffer β increment	$+0.03$ per 2,000 steps
Diffusion samples for batch argmax	100

B Details of SOLAR-Generator

B.1 Design Rationale for SOLAR

We develop SOLAR as the trajectory-based dataset for this study for three reasons.

Controlled mixed-quality trajectory generation based on realistic human error patterns. SOLAR-Generator synthesizes expert trajectories based on ARCTraj, a dataset of real human task-solving trajectories collected via O2ARC 3.0 [12]. From this data collection process, we observed that humans typically follow a correct trajectory up to an intermediate step and then suddenly deviate with an incorrect action. SOLAR-Generator models this error pattern by branching from expert trajectories at random intermediate steps and executing plausible but incorrect actions. Importantly, the injected actions are not uniformly random; instead, they are constrained to semantically plausible alternatives, such as changing only the spatial extent of a selection or choosing a different color from the set of valid colors. This design produces suboptimal trajectories that reflect realistic human mistakes rather than arbitrary noise.

Flexible multi-task training and evaluation. SOLAR provides approximately 20 task variants derived from ARC-AGI, and SOLAR-Generator can synthesize trajectories for these tasks on demand. Our experiments use 5–10 tasks, enabling controlled multi-task training and evaluation of interference effects.

Explicit action-level state transitions for stage-wise analysis. SOLAR represents task-solving trajectories at the action level, and we integrate ARCLE [9] to obtain explicit state transitions. This setup allows us to attribute performance degradation to specific stages (proposal, evaluation, selection, application) of the decision pipeline.

B.2 Operations in SOLAR

The operations from 0 to 34 are identical to those used in ARCLE [9]. Since `Submit` is an operation that receives a reward, it should only be used when the state is considered correct and not excessively. Due to LDCQ's fixed horizon, and to ensure that the agent only uses `Submit` when the state is definitively correct, we added a `None` operation that fills all subsequent states after `Submit` with the 11th color (10), which does not exist in the original ARC (0–9). In other words, during training, the `None` action emphasizes that the episode ends after `Submit`.

B.3 Grid Maker

To generate SOLAR, we develop a SOLAR-Generator that synthesizes a large volume of data for a given rule. Grid Maker is a hard-coded program specific to each task. Grid Maker contains the rules for synthesizing demonstration examples and test examples, and the synthesized solution action path consists of operations and selections. In Grid Maker, data is formatted to be compatible with ARCLE. The Grid Maker constructs analogies with the same problem semantics but with various attributes, such as the shape, color, size, and position of objects. SOLAR-Generator can generate intermediate trajectories by interacting with ARCLE. The SOLAR-Generator algorithm is designed to augment specific tasks using the Grid Maker, which can be primarily divided into three parts.

Figure 6: All operations compatible with SOLAR, 0–34 operations follow ARCLE, and only in SOLAR, 35 (`None`) is for terminated episode. This indicates that the episode ends after `Submit`.

Grid Maker was built as a data loader used in ARCLE. In the original ARCLE environment, there was no need to load operations and selections. Only the grid was loaded with the original ARC. To change this structure, the entire environment would need to be recreated. Instead, operations and selections are now loaded from the data loader's description, thereby preserving the original environment. Therefore, the process of creating input-output examples and generating action sequences works within a single file.

Specifying Common Parts Each task in the ARC dataset usually contains 3 demonstration examples, with common elements observed across these pairs. In the common parts, attributes such as color, task type, and object presence are determined by random values before pair generation.

Synthesizing Examples In the example synthesis phase, the input of the original task is augmented in a way that ensures diversity while preserving the integrity of the problem-solving method. A random input grid is generated under conditions that satisfy the task's required analogy. A solution grid is created using a hard-coded algorithm. For tasks involving pattern-based problems, as experimented in the paper, selections are made to fit the grid size, and various operations are executed either randomly or in a predetermined order. For object-based problems, the solution grid is generated by an algorithm that identifies the required objects in the input grid and processes them in accordance with the task requirements.

Converting to ARCLE Trajectories This stage involves the creation of an ARCLE-based trajectory that meticulously adheres to the problem-solving schema of the synthesized examples. The entire process is implemented using a hard-coded algorithm. During the example synthesis process, object locations may already be known, or they can be identified using a search algorithm. The information obtained is used to make appropriate selections, and the trajectory is converted into an ARCLE trajectory via an algorithm that yields the correct solution.

If all steps are properly coded, it is possible to generate the operations and selections that yield the correct solution for any randomly generated input grid. These are then fed into ARCLE to obtain intermediate states, rewards, and other information, and to verify whether the correct result is reached. Once steps 1) to 3) are correctly implemented, SOLAR-Generator can continuously and automatically generate as much data for the given task as the user desires, using the Grid Maker.

Algorithm 1: SOLAR-Generator

Input: Task set T, grid size (H, W), samples N, examples E, horizon H_{seg}
Output: Training and test trajectory datasets $\{\tau^{train}, \tau^{test}\}$

for $task \in T$ **do**
  $\mathcal{H}_{test} \leftarrow \emptyset$; `// Test set hash tracking`
  `// Generate test data first for independent evaluation`
  for $type \in \{test, train\}$ **do**
    `// Generate grids using GridMaker`
    $\mathcal{D}_s \leftarrow$ GridMaker$(task, (H, W), N_{type}, E, seed_{type})$
    for $(ex_{in}, ex_{out}, pr_{in}, pr_{out}, desc) \in \mathcal{D}_s$ **do**
      `// Duplicate detection and filtering`
      $hash \leftarrow$ SHA256$(pr_{in} || pr_{out})$
      if *duplicate or (*$type = train$ *and* $hash \in \mathcal{H}_{test}$*)* **then**
        continue
      end
      `// Execute action sequence in ARCLE environment`
      $s_0 \leftarrow$ `ARCLE.reset`(pr_{in})
      for $(op_t, sel_t) \in (desc.ops, desc.sels)$ **do**
        $s_{t+1}, r_t, done_t \leftarrow$ `ARCLE.step`(s_t, a_t)
        $\tau \leftarrow \tau \cup \{s_t, a_t, r_t, done_t\}$
      end
      `// Validate expert trajectories`
      if *is_expert and* $s_t.grid \neq pr_{out}$ **then**
        continue
      end
      `// Save trajectory (whole and segmented)`
      Save τ and $\{\tau_i^{seg}\}$ with horizon H_{seg}
    end
    if $type = test$ **then**
      $\mathcal{H}_{test} \leftarrow$ current hashes
    end
  end
end
return $\{\tau^{train}, \tau^{test}\}$