

Diffusion ReRoll: Revisable Denoising for Robotic Sequential Prediction

Seonsoo Kim Seongil Hong Jun-Gill Kang

Agency for Defense Development

{sunsoo3165, science4729, jungillkang}@gmail.com

seonsoo-pl.github.io/DiffusionReRoll

Abstract: We propose Diffusion ReRoll, a diffusion-based framework for robotic sequential prediction that enables revisable denoising over horizons. Existing diffusion-based sequence predictors typically perform a single monotonic denoising process. In contrast, Diffusion ReRoll selectively re-noises regions that have become locally stable while the remaining regions continue denoising, so the re-noised regions can be refined again using context from the rest of the horizon. This structured re-noising enables iterative cross-horizon revision, allowing earlier and later segments to revise one another, while maintaining local consistency. We evaluate Diffusion ReRoll against full-sequence diffusion and causal denoising based on Diffusion Forcing across long-horizon planning, policy learning, and unified video-action modeling. On OGBench PointMaze and AntMaze, Diffusion ReRoll achieves relative gains in average success rate of 21% over Diffusion Forcing in matched guidance-based planning and 23% over Diffuser in matched goal-inpainting. In diffusion-policy-style action prediction, Diffusion ReRoll improves average success by 56.5% relative to Diffusion Policy across different prediction horizons and history lengths on the LIBERO-10 multi-task benchmark. In unified video-action prediction, Diffusion ReRoll improves policy and inverse dynamics performance, especially under out-of-distribution evaluation, and achieves the best action-video consistency. These results support structured re-noising as an effective mechanism for revisable robotic sequence generation.

Keywords: Diffusion, Sequential Prediction, Cross-Horizon Revision

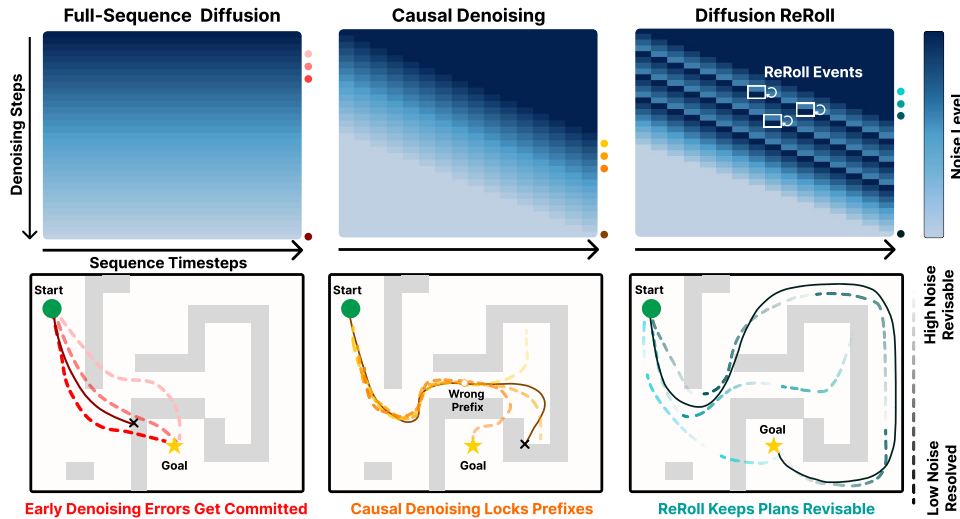


Figure 1: Comparison of full-sequence diffusion, causal denoising, and Diffusion ReRoll. ReRoll events enable cross-horizon revision, preventing premature commitment to erroneous predictions.

1 Introduction

Sequential prediction is central to robotics and embodied intelligence, including video prediction, planning, policy learning, and behavior synthesis. Diffusion models [1, 2, 3] are widely used for these tasks because they capture high-dimensional, temporally correlated, multimodal outputs [4, 5, 6, 7, 8, 9]. However, standard full-sequence diffusion denoises with a shared noise level across the entire horizon monotonically, giving limited control over temporal dependencies across the horizon.

Prior work has introduced temporal structure for diffusion-based sequential generation. Some methods compose smaller generative components for hierarchical or stitched long-horizon planning, as in CompDiffuser [10, 11]; others assign timestep-dependent noise levels, as in Diffusion Forcing, enabling flexible schedules such as causal denoising [12, 13, 14]. These approaches make sequence organization more explicit, but how distant horizon segments influence one another during denoising remains less directly addressed. For example, an early segment may enter a low-noise state before terminal constraints are incorporated, making an erroneous prefix difficult to correct later.

This motivates incorporating revisability into the generative process without discarding sequential structure. Prior revision methods revisit intermediate outputs through noise injection for editing [15], diffusion resampling for inpainting [16], masked re-prediction for iterative token generation [17, 18], and repair-based refinement for infeasible diffusion plans [19]. These works show the value of revision, but leave open how to decide which parts of a sequence should be corrected.

Rather than adding a separate repair module, we ask whether the diffusion denoising process itself can provide this mechanism. In robotic control, stochasticity and supervised iterative denoising provide beneficial inductive biases for generative policies [20, 21]. This suggests that stochasticity during training may shape the model’s learned inductive bias, rather than serving only as noise to be removed. In video diffusion, different denoising steps have been associated with exploration, correction, and consolidation [22], suggesting that noise level can influence the role a prediction plays during generation. Together, these observations suggest that structured stochastic noise-level patterns, rather than independent noise sampling, can provide a useful inductive bias for revisable denoising. These patterns assign sequence regions to noise levels with different functional roles, while the noise level scheduling controls when and where those roles are used across the horizon.

We propose *Diffusion ReRoll*, a schedule matrix-based framework for revisable robotic sequence generation. Building on per-token noise conditioning, Diffusion ReRoll organizes a single trajectory with virtual piecewise-linear noise-level patterns, called linear chunks, without explicitly splitting the trajectory. To support the desired schedule, we train with randomized linear chunks, exposing the model to structured token-wise stochasticity. During denoising, regions that reach the predefined reset trigger level are re-noised and refined again under updated context from the rest of the sequence, preventing premature commitment across the full horizon. This enables iterative cross-horizon revision, where earlier and later segments mutually revise one another before the final sequence is produced. Moreover, by changing the schedule matrix, the same model can adjust how information propagates across the horizon, emphasizing forward, backward, or endpoint-conditioned influence.

We evaluate Diffusion ReRoll across long-horizon planning, policy learning, and unified video-action modeling. On OGBench PointMaze and AntMaze [23], Diffusion ReRoll improves average success from 72.8% to 88.1% over Diffusion Forcing [14] in guidance-based replanning, and from 73.8% to 90.8% over Diffuser [6] in goal inpainting. In diffusion-policy-style action prediction on LIBERO-10 [24], ReRoll improves average success from 40.1% to 62.7% over Diffusion Policy [4] across observation histories and prediction horizons. In unified video-action prediction [25], ReRoll improves out-of-distribution (OOD) joint-policy success from 57.2% to 71.0% and reduces zero-action error, an action-video consistency proxy, from 0.324m to 0.301m.

Our contributions are threefold. First, we introduce Diffusion ReRoll, a schedule matrix-based framework with virtual linear chunks, re-noising, and linear-chunk training. Second, we present the schedule matrix as a programmable revision interface that induces desired revision and information-flow patterns without changing the model architecture. Third, we demonstrate empirical improvements on representative planning, policy-learning, and video-action prediction benchmarks.

2 Preliminaries

2.1 Generative Sequential Prediction

We consider conditional sequence generation, where a model predicts $x_{1:T} = (x_1, \dots, x_T)$ given context c . Depending on the task, x_t may denote an action, state, or video latent, while c may include observations, goals, or rewards. Throughout the paper, subscripts denote sequence timesteps and superscripts denote diffusion noise levels. Thus, x_t is the token at sequence timestep t , and x_t^k is the same token corrupted to noise level k . Here, $k \in \{0, \dots, K\}$ is a discrete diffusion index. $k = 0$ denotes a clean sample and $k = K$ denotes the Gaussian noise state. We use “noise level” to refer to this diffusion index, not the exact noise variance or signal-to-noise ratio (SNR) value.

A standard diffusion model corrupts a clean sequence $x_{1:T}^0$ as

$$x_{1:T}^k = \sqrt{\bar{\alpha}_k} x_{1:T}^0 + \sqrt{1 - \bar{\alpha}_k} \epsilon, \quad \epsilon \sim \mathcal{N}(0, I), \quad \bar{\alpha}_k = \prod_{i=1}^k (1 - \beta_i), \quad (1)$$

where $\{\beta_i\}_{i=1}^K$ is the noise variance schedule, and trains a denoising network $\epsilon_\theta(x_{1:T}^k, k, c)$ to predict the injected noise. At inference, the model starts from noise and iteratively denoises the sequence.

2.2 Per-Token Noise Levels and Diffusion Forcing

Standard full-sequence diffusion assigns one noise level to the entire sequence. Diffusion Forcing (DF) generalizes this by allowing each token to have its own noise level [14]. We denote the per-token noise levels, noisy sequence, and schedule matrix as

$$\mathbf{k} = (k_1, \dots, k_T), \quad x_{1:T}^{\mathbf{k}} = (x_1^{k_1}, \dots, x_T^{k_T}), \quad \mathbf{S} \in \{0, \dots, K\}^{M \times T}. \quad (2)$$

Here, k_t is the diffusion noise level assigned to token x_t , and $\mathbf{S}_{m,t}$ specifies the noise level of token t at denoising step row m .

This formulation can be viewed as noise-as-masking. x_t^0 is fully visible, x_t^K is effectively masked, and intermediate levels are partially visible. Per-token noise levels therefore place different sequence tokens at different denoising states. Diffusion Forcing trains with randomly assigned token-wise noise levels, which enables the same model to be sampled with structured schedule matrices, including causal denoising schedules that resolve earlier tokens before later ones. Fig. 2 visualizes the causal denoising. Diffusion ReRoll adopts this per-token noise-level view, but uses a different training distribution and denoising schedule.

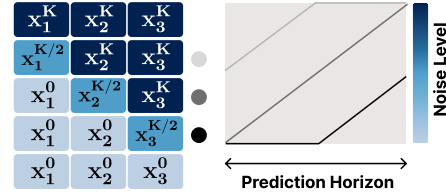


Figure 2: Causal Denoising by Per-Token Noise Levels

3 Diffusion ReRoll

Diffusion ReRoll (DR) is a diffusion-based sequential prediction framework that actively manages token-wise noise levels through a schedule matrix. The schedule matrix is defined over sequence timesteps and denoising steps, specifying the noise level assigned to each token at each denoising step. Unlike full-sequence diffusion, DR imposes piecewise-linear noise-level patterns over the sequence. Each linear region is called a *linear chunk*. These chunks are not explicit trajectory partitions; they are virtual patterns in the schedule matrix that control how different horizon regions are denoised and re-noised.

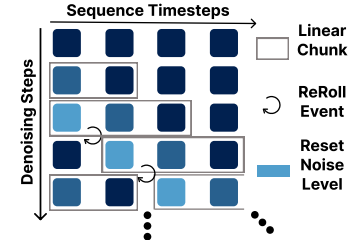


Figure 3: Part of simplified Diffusion ReRoll schedule.

During denoising, when a chunk edge reaches the reset noise level, a ReRoll event re-noises the corresponding token region and attaches it to an adjacent chunk. The schedule therefore does not

monotonically denoise the whole horizon. Instead, some regions continue denoising while selected regions return to higher noise levels for further refinement under updated context. Fig. 3 visualizes this procedure. This keeps the horizon revisable while preserving local chunk-wise structure.

DR does not require a new backbone architecture. It only requires per-token noise-level conditioning, as in Diffusion Forcing [14, 26, 27]. In our implementation, we use a transformer backbone where each sequence timestep is represented as one token, and each token receives its own noise-level embedding.

3.1 Schedule Matrix for Diffusion ReRoll

We describe denoising with a schedule matrix $\mathbf{S}$, where $S_{m,t}$ denotes the noise level of token t at schedule row m . Each transition between consecutive rows is implemented by Denoising Diffusion Implicit Models (DDIM) update [28]. While full-sequence diffusion assigns a shared noise level to all tokens, Diffusion ReRoll constructs $\mathbf{S}$ using virtual piecewise-linear chunks, which can be interpreted as denoising waves moving across the prediction horizon. Fig. 4 shows the relationship between schedule matrix and denoising waves. When the leading edge of a wave reaches the reset noise level, the corresponding token region is re-noised and becomes part of an adjacent wave. The schedule matrix is controlled by three parameters: (i) denoising slope, (ii) reset noise level, and (iii) number of ReRoll events.

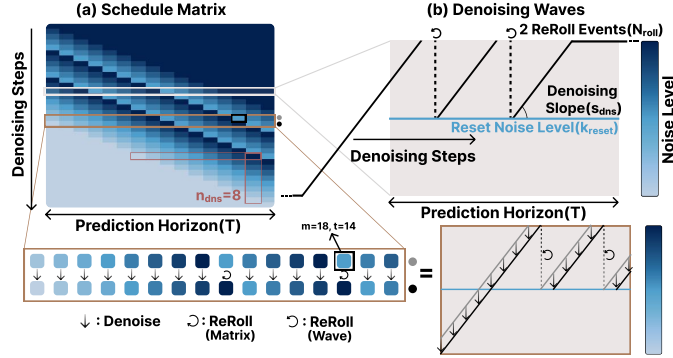


Figure 4: Representation of Diffusion ReRoll with (a) Schedule Matrix and (b) Denoising Waves.

Denoising slope. Let T be the prediction horizon, K the maximum noise level, and n_{dns} the number of DDIM rows required for one token to move from full noise K to clean value 0 within a schedule matrix. We define the denoising slope as $s_{\text{dns}} = T/n_{\text{dns}}$. For example, $T = 16$ and $n_{\text{dns}} = 8$ gives $s_{\text{dns}} = 2$ in Fig. 4. A larger s_{dns} creates shorter chunks for local refinement, while a smaller s_{dns} produces longer chunks and preserves broader temporal context.

Reset noise level. Let k_{reset} denote the reset noise level. It is the threshold that triggers a ReRoll event. When the leading edge of a chunk reaches k_{reset} , the corresponding token region is re-injected with full noise K . If k_{reset} is too high, the chunk is re-noised before it is sufficiently denoised. If it is too low, the chunk becomes nearly deterministic before revision. We find that an intermediate value, around the middle of noise level range, works robustly across noise schedules.

Number of ReRoll events. Let N_{roll} be the number of ReRoll events. This parameter controls how many times full noise is reintroduced to each token during the denoising process. Too few events limit horizon-wide revision, while too many can make re-noising redundant and increase the number of DDIM rows without further benefit. We choose N_{roll} to distribute structured revision across the horizon while keeping the denoising budget comparable to the baseline.

Bidirectional schedule matrix. The default *forward schedule matrix* propagates chunks causally toward the future. When both initial and terminal constraints are available, we use a *bidirectional schedule matrix*, where chunks start from both ends of the horizon. The two

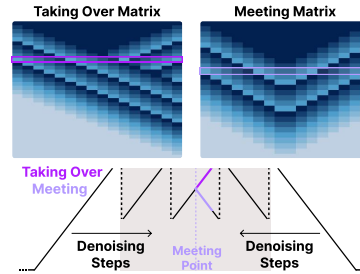


Figure 5: Two bidirectional schedule matrices are shown with denoising wave representation.

sides may use different s_{dns} , k_{reset} , and N_{roll} . Their meeting point can also be shifted to control information flow between start and terminal constraints. Since unresolved regions remain noisy and revisable, DR naturally supports non-causal and asymmetric schedule matrices.

As shown in Fig. 5, we consider two variants: a *taking over matrix*, where forward direction continues after overlap, and a *meeting matrix*, where forward and backward fronts meet at the meeting point. Both variants preserve the ReRoll principle, while using endpoint information for sequence revision differently. Details are provided in Appendix A.2.

3.2 Training with Linear Chunks

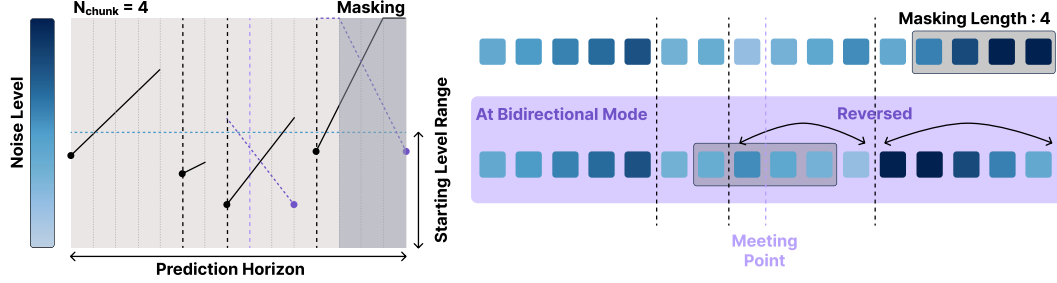


Figure 6: Training with randomized linear noise-level chunks. We sample piecewise-linear noise profiles with random chunk numbers, chunk slopes, starting noise levels, and masking length. Bidirectional mode requires additional modifications.

DR uses piecewise-linear noise-level chunks at deployment. Training only on the fixed schedule matrix would expose the model to narrow noise configurations, so we use randomized linear chunks that preserve local linear structure while covering a broader noise-level distribution.

Fig. 6 illustrates the randomized linear-chunk noise assignment. For each training sequence, we sample the number of chunks N_{chunk} and split the horizon into N_{chunk} contiguous segments. Each segment receives a starting noise level and a chunk slope, defining a linear noise profile over its tokens. Thus, token-wise noise levels are generated from structured local patterns rather than sampled independently. Let s_{dns} be the deployment denoising slope. By default, we sample $N_{\text{chunk}} \in \{1, \dots, N_{\text{max}}\}$ with $N_{\text{max}} = 2s_{\text{dns}}$, sample chunk slopes from $[0.5, 2.5]s_{\text{dns}}$, and sample chunk starting levels from a zero-to-middle noise range except for the first chunk. N_{max} , chunk slope range, and starting level range are training hyperparameters and can be adjusted for different tasks or deployment matrices.

We also apply masking to match unresolved regions at deployment. Forward training masks a random suffix with full noise. Bidirectional training masks a symmetric region around the meeting point and reverses right-side chunks, exposing the model to revision from both directions.

Given token-wise noise levels $k_{1:H}$, we train with the standard diffusion denoising loss, changing only the token-wise noise sampling distribution. Appendix B.1.1 shows that this linear-chunk training is necessary for effective ReRoll deployment. Details for training are explained in Appendix B.

4 Experiments

We evaluate DR on long-horizon planning, policy learning, and unified video-action modeling. We follow prior codebases and protocols closely [14, 29, 4, 30, 25]. We use matched architectures for Diffusion Forcing (DF) and DR. They differ only in the training noise distribution and deployment schedule matrix. DF uses the causal denoising schedule. Full-sequence baselines follow each baseline’s standard implementation. DR uses nearly shared schedule matrix parameters across the three task families to avoid task-specific schedule tuning. Full details are provided in Appendix C.

For inference, full-sequence baselines use their standard samplers, while DF and DR use nearly matched schedule matrix DDIM samplers with no larger update counts than full-sequence baselines.

4.1 Effect of ReRoll Events

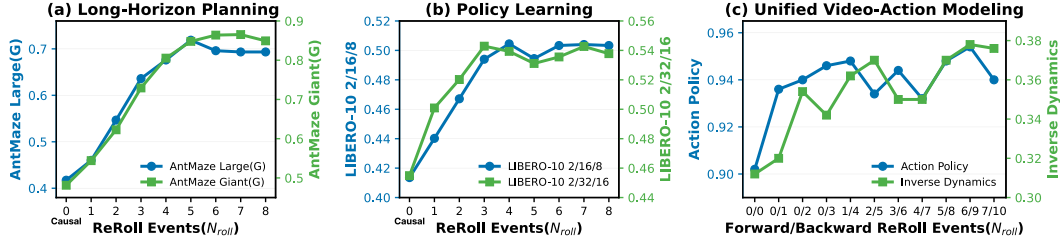


Figure 7: Effect of ReRoll events in (a) long-horizon planning, (b) policy learning, and (c) unified video-action modeling. All plotted values are average success rates shown as fractions. In (a), (G) denotes guidance-based planning; in (b), labels follow Obs./Hor./Act. and the official protocol is applied. Increasing ReRoll events generally improves performance before saturation.

Before comparing DR with other methods, we first isolate the effect of ReRoll events. With zero events, the re-noising operation is removed, and forward schedules reduce to causal denoising. Fig. 7 shows that adding ReRoll events generally improves planning, policy learning, and unified video-action modeling, before performance saturates. This indicates that DR’s gains are largely driven by selective re-noising, which enables repeated revision during denoising.

The number of ReRoll events is only one component of the schedule matrix. The denoising slope and reset noise level also shape how revision is distributed across the horizon. Additional schedule studies are provided in Appendix A. For controlled comparison, the main experiments use a default six-event schedule, with six forward events for bidirectional matrices.

4.2 Long-Horizon Planning

We evaluate Diffusion ReRoll on OGBench PointMaze and AntMaze [23]. This benchmark evaluates whether DR can maintain trajectory consistency while generating long-horizon plans that reach distant goals. We compare against Diffuser as a full-sequence diffusion baseline and Diffusion Forcing as a causal denoising baseline [6, 14]. We largely follow the MCTD planning setup [29]. Additional details are provided in Appendix C.2.

We consider two protocols, training separate models for each method and protocol. In *guidance-based replanning*, trajectories are sampled with distance-based goal guidance [31, 32] and replanning. In *goal inpainting*, the start and goal states are fixed and the model generates the intermediate trajectory. For controlled comparison, we train an inpainting-compatible DF variant using the same masking scheme and bidirectional schedule matrix as DR, but without ReRoll events.

Mode	Method	AntMaze			PointMaze		
		Giant	Large	Medium	Giant	Large	Medium
Guidance Replan	Diffuser	16.0 $\pm$ 1.1	42.1 $\pm$ 4.2	69.1 $\pm$ 1.0	26.5 $\pm$ 4.9	30.0 $\pm$ 1.8	46.4 $\pm$ 0.8
	Forcing	<u>49.1</u> $\pm$ 9.1	<u>55.3</u> $\pm$ 5.7	<u>93.9</u> $\pm$ 2.4	<u>88.9</u> $\pm$ 4.0	<u>60.8</u> $\pm$ 1.7	<u>88.7</u> $\pm$ 7.6
	ReRoll	86.4 $\pm$ 1.4	69.6 $\pm$ 1.4	98.0 $\pm$ 0.4	99.3 $\pm$ 0.8	76.3 $\pm$ 1.8	98.8 $\pm$ 0.4
Inpainting	Diffuser	33.2 $\pm$ 1.4	68.1 $\pm$ 4.3	88.3 $\pm$ 1.6	64.9 $\pm$ 1.6	96.1 $\pm$ 0.5	92.0 $\pm$ 3.9
	Forcing	<u>71.9</u> $\pm$ 4.0	67.5 $\pm$ 0.6	<u>90.5</u> $\pm$ 0.6	<u>94.5</u> $\pm$ 1.6	95.7 $\pm$ 1.8	91.2 $\pm$ 2.8
	ReRoll	80.9 $\pm$ 3.6	74.4 $\pm$ 0.4	93.7 $\pm$ 0.6	99.1 $\pm$ 0.2	99.5 $\pm$ 0.6	96.9 $\pm$ 2.7

Table 1: Long-horizon planning success rates (%) on OGBench PointMaze and AntMaze. Values are mean $\pm$ standard deviation over three training seeds. Best means are bolded and second-best means are underlined.

Table 1 shows that DR achieves the best performance under both protocols across all maze variants. The gains are largest in the Large and Giant mazes, where long-horizon consistency is most critical. Fig. 8 illustrates the mechanism. Full-sequence diffusion can commit to an early incorrect plan, while ReRoll keeps parts of the trajectory revisable and recovers a valid path.

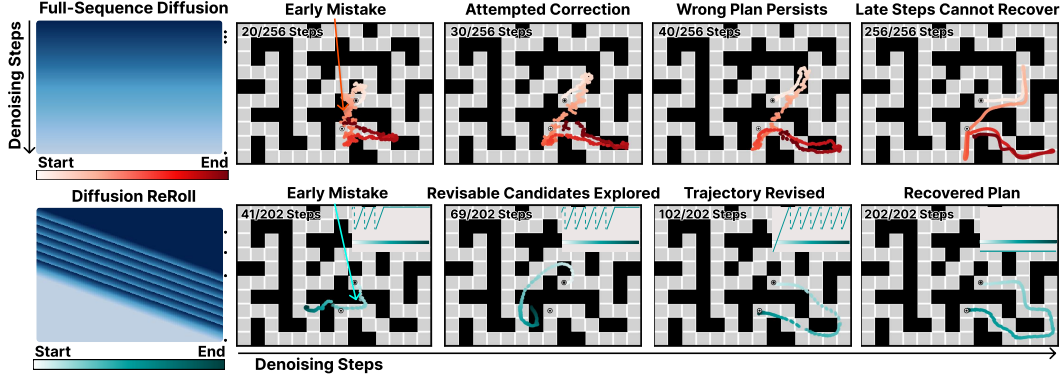


Figure 8: Denoising process of Diffuser and Diffusion ReRoll are shown. Diffusion ReRoll keeps trajectories revisable, enabling recovery from early planning errors. Goal-directed guidance is used for planning on AntMaze Giant. Predicted clean trajectories $\hat{x}_0$ are visualized.

4.3 Policy Learning

We next evaluate DR in diffusion-policy-style action prediction on LIBERO-10 multi-task [24] and RoboCasa [33] single-task learning. This tests whether DR improves action-chunk prediction across observation histories and prediction horizons. For DF and DR, we minimally modify the Chi-Transformer [4] so each action token receives its own noise level, while using each method’s action representation. LIBERO-10 policies are trained from scratch, while RoboCasa uses a pretrained vision encoder. Other settings are shared whenever possible. Details are in Appendix C.3.1.

We found that LIBERO-10 evaluation is sensitive to the reset behavior during object spawning, particularly whether the gripper remains fixed or is opened while objects settle. Opening the gripper makes the initial state closer to the training demonstration distribution. Because existing codebases differ in this initialization protocol, we report results under both the official initialization and an open-gripper initialization. Details and evaluation concerns are provided in Appendix C.3.2.

Method	LIBERO-10 Official (Obs./Hor./Act.)					LIBERO-10 Open (Obs./Hor./Act.)				
	2/16/8	2/32/16	2/64/16	4/32/16	8/32/16	2/16/8	2/32/16	2/64/16	4/32/16	8/32/16
Diff. Policy	28.1 $\pm$ 3.8	41.1 $\pm$ 8.2	35.1 $\pm$ 2.7	42.2 $\pm$ 8.8	53.8 $\pm$ 0.8	76.2 $\pm$ 0.5	65.5 $\pm$ 1.9	57.9 $\pm$ 2.0	69.1 $\pm$ 1.4	64.7 $\pm$ 1.9
Forcing	<u>37.3</u> $\pm$ 12.0	<u>41.6</u> $\pm$ 10.9	<u>41.5</u> $\pm$ 4.1	<u>63.0</u> $\pm$ 2.7	<u>62.2</u> $\pm$ 4.0	68.3 $\pm$ 1.7	60.1 $\pm$ 1.1	<u>59.0</u> $\pm$ 2.9	<u>70.7</u> $\pm$ 0.8	<u>75.1</u> $\pm$ 2.4
ReRoll	51.0 $\pm$ 24.1	56.4 $\pm$ 2.8	63.4 $\pm$ 6.4	74.5 $\pm$ 2.6	68.1 $\pm$ 7.7	82.5 $\pm$ 3.6	75.6 $\pm$ 3.1	70.7 $\pm$ 1.0	81.5 $\pm$ 2.9	82.9 $\pm$ 0.8

Table 2: LIBERO-10 multi-task policy learning results under two initialization protocols. We report success rates (%) as mean $\pm$ standard deviation over three training seeds, using the best checkpoint from each run. Best mean results are bolded and second-best mean results are underlined.

Method	CloseSingleDoor			CoffeePressButton			CoffeeServeMug			PnPCounterToSink (300 Demos)		
	2/32/16	2/64/16	4/32/16	2/32/16	2/64/16	4/32/16	2/32/16	2/64/16	4/32/16	2/32/16	2/64/16	4/32/16
Diff. Policy	52.0 $\pm$ 2.0	52.7 $\pm$ 2.3	<u>56.7</u> $\pm$ 8.3	76.0 $\pm$ 5.3	54.7 $\pm$ 8.3	80.0 $\pm$ 4.0	43.3 $\pm$ 3.1	32.7 $\pm$ 4.2	39.3 $\pm$ 5.8	30.7 $\pm$ 3.1	<u>22.0</u> $\pm$ 4.0	24.7 $\pm$ 2.3
Forcing	58.7 $\pm$ 8.1	68.0 $\pm$ 3.5	65.3 $\pm$ 4.6	<u>79.3</u> $\pm$ 8.1	<u>70.7</u> $\pm$ 1.2	88.7 $\pm$ 5.0	34.0 $\pm$ 7.2	41.3 $\pm$ 4.2	<u>36.0</u> $\pm$ 5.3	<u>28.7</u> $\pm$ 4.3	29.3 $\pm$ 1.2	31.3 $\pm$ 6.1
ReRoll	67.3 $\pm$ 5.8	<u>62.0</u> $\pm$ 5.3	65.3 $\pm$ 6.4	84.0 $\pm$ 5.3	80.7 $\pm$ 7.0	<u>86.0</u> $\pm$ 4.0	<u>40.0</u> $\pm$ 5.3	<u>34.7</u> $\pm$ 4.6	<u>36.0</u> $\pm$ 3.5	<u>28.7</u> $\pm$ 3.1	29.3 $\pm$ 6.1	<u>29.3</u> $\pm$ 2.3

Table 3: RoboCasa single-task policy learning results. We report success rates (%) as mean $\pm$ standard deviation over three training seeds. Best mean results are bolded and second-best mean results are underlined.

Table 2 shows that DR achieves the best performance across LIBERO-10 settings under both initialization protocols. The gains are strongest in the multi-task setting, where diverse objects, goals, and temporal contexts make continued cross-horizon revision useful. Table 3 shows that DR is less consistently best on RoboCasa single-task learning. We hypothesize that single-task datasets contain narrower behavior distributions, leaving less room for repeated revision to resolve competing action structures. This is consistent with Sec. 4.4, where DR performs strongly after broad video-action pretraining followed by single-task fine-tuning.

Method	Act. Pol.↑ (%)	Joint Pol.↑ (%)	Inv. Dyn.↑ (%)	OOD Act.↑ (%)	OOD Joint↑ (%)	Zero-Act. Err.↓ (m)
UWM	84.4	85.6	24.0	52.8	57.2	<u>0.324</u>
UWM + DF	<u>90.0</u>	<u>86.4</u>	<u>30.6</u>	<u>58.6</u>	<u>57.6</u>	0.331
UWM + DR	95.4	94.2	37.8	67.6	71.0	0.301

Table 4: Averaged unified world model prediction results on LIBERO-10. Best results are bolded and second-best results are underlined.

4.4 Unified Video-Action Modeling

We next evaluate DR in a unified video-action prediction setting. Unified World Models (UWM) support action prediction, future-visual prediction, and joint video-action generation within a single model [25]. Here, we test whether DR improves policy generation, inverse dynamics, OOD robustness, and action-video consistency. Although such models can exploit video-only data without action annotations [8, 34, 35], our focus is downstream control performance and consistency.

We apply DR to UWM by assigning structured linear noise levels over a joint sequence of 32 action tokens and 2 future image latent tokens. Models are pretrained on LIBERO-90 and fine-tuned on each LIBERO-10 task. We compare UWM, UWM + DF, and UWM + DR under action-only policy prediction, joint video-action policy prediction, inverse dynamics, and OOD evaluation. In the OOD setting, task-irrelevant background objects are removed and object positions are randomized. Details are provided in Appendix C.4.1.

Table 4 shows that DR improves policy prediction and inverse dynamics, with the largest gains under visual OOD evaluation. Additional analyses of schedule matrix applications are provided in Appendix A.2.2.

We further test action-video consistency with a zero-action intervention. Since task-finetuned models are trained on successful behaviors, zero-actions are OOD. We fix future action tokens to zero, generate future images, and apply inverse dynamics to measure the implied terminal end-effector motion, as shown in Fig. 9. DR achieves the lowest zero-action error, indicating stronger consistency between imposed actions and generated visual dynamics.

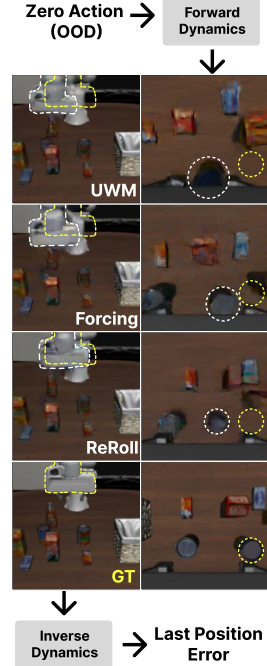


Figure 9: Procedure of zero action experiment and generated last scene images are visualized.

5 Limitations

This work has several limitations. First, all experiments are in simulation, and Diffusion ReRoll has not yet been validated on real robots. Second, ReRoll still requires schedule tuning for new tasks. Its denoising slope, reset noise level, and number of ReRoll events affect both performance and inference speed. Although we use a shared default matrix and provide schedule studies, selecting these parameters remains task-dependent. Third, ReRoll is tested with transformer denoisers and benchmark-scale datasets. Validation in general robot models or larger VLAs remains future work.

6 Conclusion

We introduced Diffusion ReRoll, a schedule matrix-based framework for revisable robotic sequence generation. By organizing trajectories into virtual linear noise-level chunks and selectively re-noising resolved regions, ReRoll uses denoising itself as a revision interface. This allows horizon segments to remain revisable across a prolonged denoising process. Across planning, policy learning, and unified world modeling benchmarks, ReRoll improves long-horizon recovery, robust action prediction, and action-video consistency, suggesting that diffusion schedules can serve as controllable mechanisms for robotic sequence generation.

Acknowledgments

This work was supported by the Agency for Defense Development grant funded by the Korean Government in 2026.

References

- [1] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. pmlr, 2015.
- [2] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.
- [3] A. Ajay, Y. Du, A. Gupta, J. Tenenbaum, T. Jaakkola, and P. Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.
- [4] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.
- [5] J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet. Video diffusion models. *Advances in neural information processing systems*, 35:8633–8646, 2022.
- [6] M. Janner, Y. Du, J. Tenenbaum, and S. Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, 2022.
- [7] S. Li, Y. Gao, D. Sadigh, and S. Song. Unified video action model. *arXiv preprint arXiv:2503.00200*, 2025.
- [8] B. Chen, T. Zhang, H. Geng, C. Zhang, P. Li, K. Song, W. T. Freeman, J. Malik, P. Abbeel, R. Tedrake, V. Sitzmann, and Y. Du. Large video planner enables generalizable robot control. *arXiv preprint arXiv:2512.15840*, 2025.
- [9] U. A. Mishra, S. Xue, Y. Chen, and D. Xu. Generative skill chaining: Long-horizon skill planning with diffusion models. In *7th Annual Conference on Robot Learning*, 2023.
- [10] Y. Luo, U. A. Mishra, Y. Du, and D. Xu. Generative trajectory stitching through diffusion composition. *arXiv preprint arXiv:2503.05153*, 2025.
- [11] S. Kim, Y. Choi, D. E. Matsunaga, and K.-E. Kim. Stitching sub-trajectories with conditional diffusion model for goal-conditioned offline rl. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 13160–13167, 2024.
- [12] T. Wu, Z. Fan, X. Liu, H.-T. Zheng, Y. Gong, J. Jiao, J. Li, J. Guo, N. Duan, W. Chen, et al. Ar-diffusion: Auto-regressive diffusion model for text generation. *Advances in Neural Information Processing Systems*, 36:39957–39974, 2023.
- [13] D. Ruhe, J. Heek, T. Salimans, and E. Hogeboom. Rolling diffusion models. *arXiv preprint arXiv:2402.09470*, 2024.
- [14] B. Chen, D. Martí Monsó, Y. Du, M. Simchowitz, R. Tedrake, and V. Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2024.
- [15] C. Meng, Y. He, Y. Song, J. Song, J. Wu, J.-Y. Zhu, and S. Ermon. SDEdit: Guided image synthesis and editing with stochastic differential equations. In *International Conference on Learning Representations*, 2022.

- [16] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. Van Gool. Repaint: In-painting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11461–11471, 2022.
- [17] H. Chang, H. Zhang, L. Jiang, C. Liu, and W. T. Freeman. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11315–11325, 2022.
- [18] L. Zhuang, S. Fan, F. P. Audonnet, Y. Ru, E. S. Ho, G. A. Camarasa, and P. Henderson. Masked generative policy for robotic control. *arXiv preprint arXiv:2512.09101*, 2025.
- [19] K. Lee, S. Kim, and J. Choi. Refining diffusion planner for reliable behavior synthesis by automatic detection of infeasible plans. *Advances in Neural Information Processing Systems*, 36:24223–24246, 2023.
- [20] C. Pan, G. Anantharaman, N.-C. Huang, C. Jin, D. Pfrommer, C. Yuan, F. Permenter, G. Qu, N. Boffi, G. Shi, and M. Simchowitz. Much ado about noising: Dispelling the myths of generative robotic control. *arXiv preprint arXiv:2512.01809*, 2025.
- [21] D. Pfrommer, Z. Dou, C. Scarvelis, M. Simchowitz, and A. Jadbabaie. Is your diffusion model actually denoising? *Advances in Neural Information Processing Systems*, 38:94380–94420, 2026.
- [22] R. Wang, Z. Cai, F. Pu, J. Xu, W. Yin, M. Wang, R. Ji, C. Gu, B. Li, Z. Huang, H. Deng, D. Lin, Z. Liu, and L. Yang. Demystifying video reasoning. *arXiv preprint arXiv:2603.16870*, 2026.
- [23] S. Park, K. Frans, B. Eysenbach, and S. Levine. Ogbench: Benchmarking offline goal-conditioned rl. In *International Conference on Learning Representations (ICLR)*, 2025.
- [24] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.
- [25] C. Zhu, R. Yu, S. Feng, B. Burchfiel, P. Shah, and A. Gupta. Unified world models: Coupling video and action diffusion for pretraining on large robotic datasets. In *Proceedings of Robotics: Science and Systems (RSS)*, 2025.
- [26] K. Song, B. Chen, M. Simchowitz, Y. Du, R. Tedrake, and V. Sitzmann. History-guided video diffusion. *arXiv preprint arXiv:2502.06764*, 2025.
- [27] T. Yin, Q. Zhang, R. Zhang, W. T. Freeman, F. Durand, E. Shechtman, and X. Huang. From slow bidirectional to fast autoregressive video diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22963–22974, 2025.
- [28] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.
- [29] J. Yoon, H. Cho, D. Baek, Y. Bengio, and S. Ahn. Monte carlo tree diffusion for system 2 planning. *arXiv preprint arXiv:2502.07202*, 2025.
- [30] C. Liu, X. Han, J. Gao, Y. Zhao, H. Chen, and Y. Du. Oat: Ordered action tokenization. In *Proceedings of Robotics: Science and Systems*, 2026.
- [31] J. Ho and T. Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.
- [32] P. Dhariwal and A. Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

- [33] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. In *Robotics: Science and Systems (RSS)*, 2024.
- [34] T. Wiedemer, Y. Li, P. Vicol, S. S. Gu, N. Matarese, K. Swersky, B. Kim, P. Jaini, and R. Geirhos. Video models are zero-shot learners and reasoners. *arXiv preprint arXiv:2509.20328*, 2025.
- [35] J. Liang, P. Tokmakov, R. Liu, S. Sudhakar, P. Shah, R. Ambrus, and C. Vondrick. Video generators are robot policies. *arXiv preprint arXiv:2508.00795*, 2025.
- [36] T. Hang, S. Gu, C. Li, J. Bao, D. Chen, H. Hu, X. Geng, and B. Guo. Efficient diffusion training via min-snr weighting strategy. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7441–7451, 2023.
- [37] C. Chi, Z. Xu, C. Pan, E. Cousineau, B. Burchfiel, S. Feng, R. Tedrake, and S. Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.
- [38] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. Pi zero: a vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.
- [39] C. Chen, J. Baek, F. Deng, K. Kawaguchi, C. Gulcehre, and S. Ahn. Plandq: Hierarchical plan orchestration via d-conductor and q-performer. *arXiv preprint arXiv:2406.06793*, 2024.
- [40] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.
- [41] A. Mandlekar, S. Nasiriany, B. Wen, I. Akinola, Y. Narang, L. Fan, Y. Zhu, and D. Fox. Mimicgen: A data generation system for scalable robot learning using human demonstrations. In *7th Annual Conference on Robot Learning*, 2023.

Appendix

A Choosing the Schedule Matrix

A.1 DDIM Updates with Schedule Matrices

This section describes the schedule-matrix samplers used by Diffusion Forcing and Diffusion ReRoll. The schedule matrix stores discrete noise levels, not raw diffusion timestep indices. Let $\mathbf{S}_{m,i} \in \{0, \dots, K\}$ be the schedule noise level of token i at row m , where level 0 denotes the clean state and level K denotes the maximum-noise state used by the sampler. Before applying DDIM, each schedule noise level is mapped to a real diffusion timestep index by

$$\tau : \{0, \dots, K\} \rightarrow \{-1, 0, \dots, K_{\text{diff}} - 1\},$$

where K_{diff} is the number of training diffusion steps. We set $\tau(0) = -1$ for the clean level, with $\bar{\alpha}_{\tau(0)} = 1$, and $\tau(K) = K_{\text{diff}} - 1$ for the maximum-noise level.

For a transition from row m to row $m + 1$, let

$$k = \mathbf{S}_{m,i}, \quad \ell = \mathbf{S}_{m+1,i},$$

and define $a_k = \bar{\alpha}_{\tau(k)}$ and $a_\ell = \bar{\alpha}_{\tau(\ell)}$. For $k > 0$, under our x_0 -prediction parameterization, the denoising network predicts a clean token $\hat{x}_i^0$ from x_i^k , and the corresponding predicted noise is

$$\hat{\epsilon}_i = \frac{x_i^k - \sqrt{a_k} \hat{x}_i^0}{\sqrt{1 - a_k}}. \quad (3)$$

For a denoising transition $\ell < k$, we apply the DDIM update

$$x_i^\ell = \sqrt{a_\ell} \hat{x}_i^0 + \sqrt{1 - a_\ell - \sigma_{k \rightarrow \ell}^2} \hat{\epsilon}_i + \sigma_{k \rightarrow \ell} z, \quad z \sim \mathcal{N}(0, I), \quad (4)$$

where

$$\sigma_{k \rightarrow \ell} = \eta \sqrt{\frac{1 - a_k/a_\ell}{1 - a_k} (1 - a_\ell)}. \quad (5)$$

In our experiments, $\eta = 0$, so downward DDIM denoising transitions are deterministic and do not inject additional noise.

The schedule matrix can also keep or increase a token’s noise level. If $\ell = k$, the token value is left unchanged. If $\ell > k$, we treat the transition as a ReRoll reset rather than a reverse DDIM update, and apply the forward noising transition

$$x_i^\ell = \sqrt{\frac{a_\ell}{a_k}} x_i^k + \sqrt{1 - \frac{a_\ell}{a_k}} \epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \quad (6)$$

Thus, downward transitions use standard DDIM updates, while reset transitions use explicit forward noising. Even with $\eta = 0$, reset transitions remain stochastic because new Gaussian noise is injected during re-noising. This allows different tokens to denoise asynchronously while selected regions are returned to higher noise levels according to the schedule matrix.

For guidance-based sampling, the guidance term modifies the model-predicted direction before the DDIM update. The schedule-row transition rule itself remains unchanged.

A.2 Bidirectional Schedule Matrices

Diffusion ReRoll can use different schedule matrices depending on the structure of the task. When both initial and terminal constraints are available, such as goal inpainting or future-image conditioning, it is useful to introduce denoising waves from both ends of the horizon. We consider two bidirectional variants: the *meeting matrix* and the *taking over matrix*.

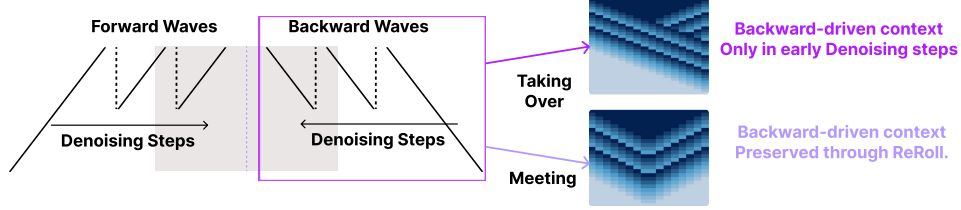


Figure 10: Bidirectional variants introduce backward waves from the terminal. The taking over matrix uses goal-side derived context early and then returns to forward revision, while the meeting matrix preserves bidirectional revision.

The *meeting matrix* introduces forward and backward waves that repeatedly revise the sequence from both directions. This creates a strongly non-causal denoising process, where non-causal information from the terminal side can continuously influence earlier segments. Such behavior might be beneficial at some tasks. However, performance can degrade on tasks where causal deployment structure is important.

The *taking over matrix* uses terminal information more conservatively. Backward waves are used in the early part of denoising to provide goal-side context, but they are later overwritten by forward causal waves. This is useful for some tasks, where persistent non-causal influence from the future can introduce an undesirable deployment bias. Thus, the taking over matrix allows endpoint information to guide early revision while preserving a more causal final prediction structure.

These variants show that the ReRoll schedule matrix is not a fixed schedule, but a design interface for controlling how information flows across the horizon. We evaluate their use in two settings below.

A.2.1 Applications to Maze Planning

For maze planning, the bidirectional schedule matrix is mainly used in the goal-inpainting setting, where both start and goal states are given. In the main experiments, we use the *taking over matrix*. The meeting and taking over matrices show similar overall performance, but the taking over matrix is slightly better when execution is restricted to a planned prefix.

Although the main bidirectional maze experiments use goal inpainting, the same idea can also be applied to guidance-based replanning. Table 5 compares the default forward ReRoll matrix with a bidirectional taking over matrix under the guidance + replanning protocol. The result suggests that endpoint-aware revision can improve some long-horizon cases, especially AntMaze Large, while preserving competitive performance on other tasks. This illustrates that ReRoll schedule matrices can control information flow between causal execution and terminal constraints through selective re-noising.

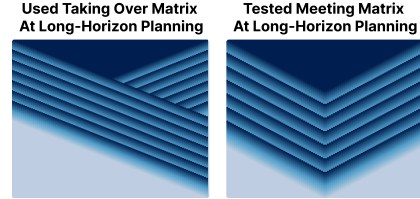


Figure 11: Bidirectional schedule matrices tested on OGBench maze planning. The taking over matrix performs slightly better.

Task	Forward matrix	Bidirectional taking over
AntMaze Giant	86.4 $\pm$ 1.4	87.2
AntMaze Large	69.6 $\pm$ 1.4	77.6
AntMaze Medium	98.0 $\pm$ 0.4	97.6
PointMaze Giant	99.3 $\pm$ 0.8	100.0
PointMaze Large	76.3 $\pm$ 1.8	74.4
PointMaze Medium	98.8 $\pm$ 0.4	100.0

Table 5: Forward and bidirectional ReRoll schedules under guidance-based replanning. Forward-matrix results are mean $\pm$ standard deviation over three training seeds. Bidirectional taking over results are single-seed and intended as exploratory evidence of schedule-matrix flexibility.

A.2.2 Applications to Unified Video-Action Modeling

We also compare bidirectional schedule matrices in Unified Video-Action Modeling, where action tokens and future-image tokens are denoised in a joint sequence. Unlike maze planning, the terminal side contains future visual tokens, which can provide useful context for action generation. We find that the *meeting matrix* consistently outperforms the *taking over matrix*, suggesting that preserving bidirectional interaction between action and image tokens is beneficial for video-action modeling. The taking over matrix can let action-side context dominate later image generation, which may reduce the benefit of future visual context.

We place the meeting point at 0.765 of the joint action-image horizon rather than at the midpoint. This biases the schedule toward image-to-action revision while limiting unnecessary non-causal action-to-action influence.

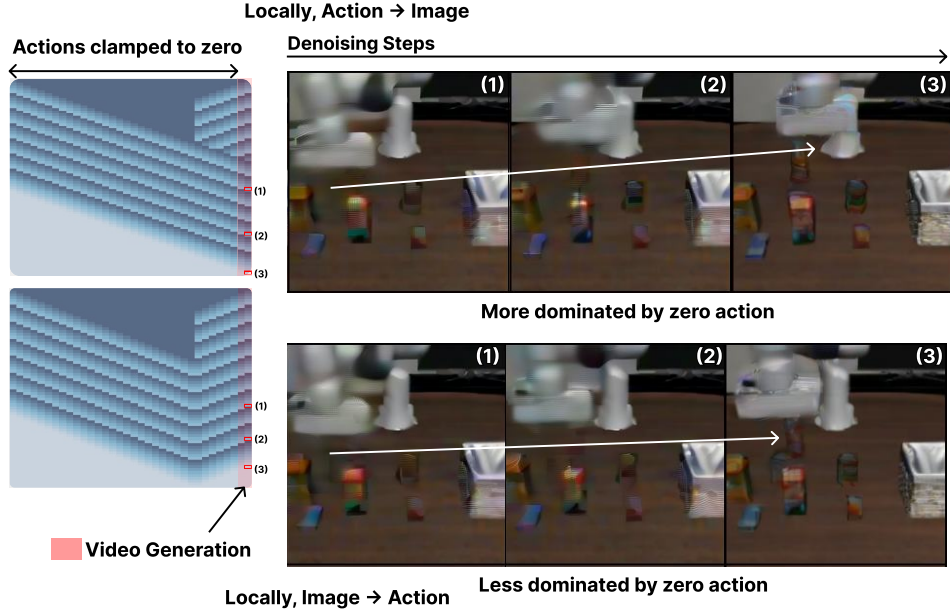


Figure 12: Zero-action visualization for Unified Video-Action Modeling schedule matrices. Action tokens are clamped to zero while future image tokens are generated according to each schedule matrix. The taking over matrix is more strongly influenced by the zero-action condition, whereas the meeting matrix is less influenced. Predicted clean latents $\hat{x}_0$ are visualized.

Fig. 12 further visualizes how the two matrices induce different cross-modal information flow. This diagnostic is separate from the quantitative zero-action intervention in the main experiments. Here, we clamp the action-token values to zero but keep their noise levels following the schedule matrix, and then generate the future image tokens. With the taking over matrix, later denoising is more locally action-to-image, so the generated video is strongly pulled toward the zero-action behavior. With the meeting matrix, image-side context remains active through bidirectional revision, making the generated future images less dominated by the zero-action condition. This illustrates that ReRoll schedule matrices can control whether information flows primarily from actions to images or from images back to actions during denoising.

A.3 ReRoll Schedule Ablations

The denoising slope controls the temporal span and propagation speed of each denoising wave. The reset noise level determines when a wave is restarted. The number of ReRoll events determines how broadly repeated revision is distributed across the horizon. Although the main experiments use nearly shared default settings for controlled comparison, the appropriate schedule can depend on horizon length, temporal information density, desired chunk granularity, and inference budget. The following experiments illustrate how these choices affect performance and guide schedule selection.

A.3.1 Same ReRoll Parameters at Different Horizons

The ReRoll schedule matrix is controlled by three coupled parameters: denoising slope, reset noise level, and number of ReRoll events. Because the schedule matrix is defined over both sequence timesteps and DDIM rows, these parameters have direct meaning with respect to the prediction horizon. The same normalized parameter values at different horizon lengths can induce similar revision behavior across tasks, but the actual number of DDIM updates and the available noise-level resolution depend on the horizon length.

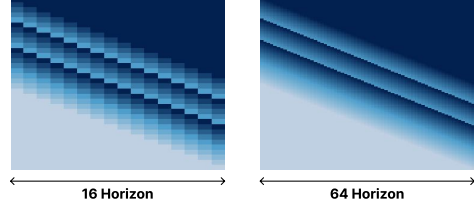


Figure 13: Schedule matrices with same ReRoll parameters under different horizons.

For example, with a short horizon, the reset noise level can only be placed at coarse discrete positions in the matrix. With horizon $T = 16$ and denoising slope $s_{\text{dns}} = 4$, only a small set of normalized reset levels, such as 0.75, 0.5, 0.25, and 0, can be represented. Longer horizons provide finer resolution and allow the same parameterization to realize smoother schedule variations.

A.3.2 Effect of Reset Noise Level

The reset noise level and the number of ReRoll events jointly control how much revision is applied by the schedule matrix. The reset noise level determines when an active denoising wave is restarted. If the reset level is too high, a region is re-noised before it is sufficiently denoised; if it is too low, the region becomes nearly deterministic before being revisited.

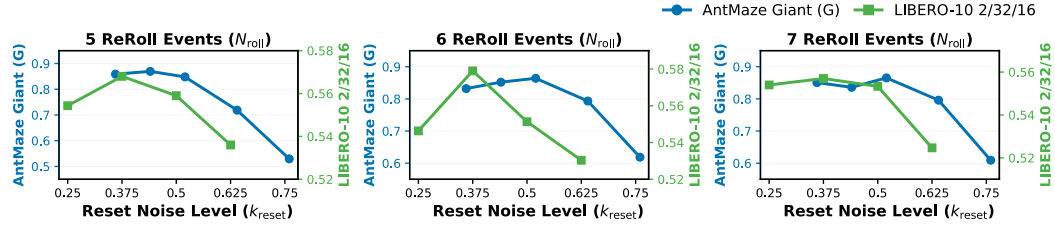


Figure 14: Reset noise-level ablation under different numbers of ReRoll events. Performance is best at intermediate reset levels and degrades when the reset level is too high or too low. The official protocol is applied at LIBERO-10.

Fig. 14 ablates the reset noise level on AntMaze Giant guidance planning and LIBERO-10 Official 2/32/16. Across different ReRoll counts, both tasks generally perform best at intermediate reset levels, while high reset levels often degrade performance because chunks are restarted before enough denoised structure is formed. The best reset level differs across tasks; for example, LIBERO-10 can benefit from a lower reset level than the $k_{\text{reset}} = 0.5K$ setting used in the main experiments, whereas AntMaze Giant achieves better performance at $k_{\text{reset}} = 0.52K$. This likely reflects differences in task horizon, temporal density, and the underlying noise schedule described in Appendix C.

A.3.3 Results with Lower Denoising Slopes

Several ReRoll training and schedule-matrix parameters are tied to the denoising slope s_{dns} . While the main experiments mostly use $s_{\text{dns}} = 4$ or nearby values, lower slopes can also work well when the training chunk distribution is adjusted accordingly. For common prediction horizons used in our experiments, we recommend starting with $s_{\text{dns}} \in [2, 4]$ and then adjusting the chunk slope range, reset level, and number of ReRoll events according to the task horizon and inference budget. We include representative lower-slope configurations in Table 6. These results are not intended as a direct comparison, but they show that ReRoll can be trained and deployed with different schedule-matrix parameterizations across tasks.

Task	Schedule matrix			Training chunks			Setting	Success rate	
	s_{dns}	k_{reset}	N_{roll}	Slope range	Start range	N_{max}		Main / Official	Open
Default schedule used in main experiments									
AntMaze Large	4	0.52	6	[2, 10]	[0, 0.5K]	8	Guidance	69.6 ± 1.4	—
AntMaze Giant	4	0.52	6	[2, 10]	[0, 0.5K]	8	Guidance	86.4 ± 1.4	—
LIBERO-10	4	0.50	6	[2, 10]	[0, 0.5K]	8	2/16/8	51.0 ± 24.1	82.5 ± 3.6
Lower-slope example configurations									
AntMaze Large	1	0.52	2	[0.5, 2]	[0, 0.5K]	4	Guidance	66.4	—
AntMaze Large	2	0.52	3	[1, 4]	[0, 0.5K]	6	Guidance	69.2	—
AntMaze Giant	2	0.52	3	[1, 4]	[0, 0.5K]	6	Guidance	74.0	—
LIBERO-10	2	0.50	2	[1, 4.5]	[0, 0.5K]	4	2/16/8	47.2 ± 13.2	77.0 ± 2.8

Table 6: Example ReRoll schedule and linear-chunk training configurations. Default settings are used in the main experiments, while lower-slope settings show that ReRoll can also be trained and deployed with different schedule-matrix parameters. Success rates are reported in percent.

A.4 Denoising Step Numbers and Inference Time

The inference cost of DR is determined by the schedule matrix. Each row of the schedule matrix specifies token-wise noise levels, and one DDIM update is applied between consecutive rows. Thus, the number of DDIM updates is the number of schedule rows minus one. Unlike full-sequence diffusion, where the DDIM budget is usually chosen directly, the DR update count is induced by the prediction horizon and the schedule parameters, including denoising slope, reset noise level, and number of ReRoll events.

Longer prediction horizons generally require more schedule rows to preserve the same denoising-wave shape. Therefore, when the prediction horizon is a design choice, it should be selected together with the inference-time budget. After training with compatible linear-chunk noise patterns, the simplest way to reduce inference cost is often to use fewer ReRoll events at deployment. For example, in the policy-learning ablation in Fig. 7, performance already saturates after a moderate number of ReRoll events, so a smaller-event schedule can be used when faster inference is preferred. The reset noise level can also change the number of schedule rows, but we mainly report the effect of changing ReRoll events. For visualization of these matrices, see Appendix C.

N_{roll}	Planning	Policy learning			Unified Video-Action
		16/8	32/16	64/16	
0	124	11	23	31	34
1	137	14	28	40	40
2	150	17	33	49	46
3	163	20	38	58	52
4	176	23	43	67	58
5	189	26	48	76	64
6	202	29	53	85	70
7	215	32	58	94	76
8	228	35	63	103	82

Table 7: DDIM update counts induced by ReRoll schedule matrices. For video-action generation, N_{roll} denotes the forward-side ReRoll count. 16/8 denotes prediction horizon/action steps.

These counts show that schedule design directly affects inference cost. Increasing ReRoll events generally improves revisability but adds DDIM updates, while fewer events can reduce latency when performance has already saturated. In practice, we choose the schedule by considering both task performance and the available inference budget.

Policy inference time. We additionally measure real full-policy inference latency on a single NVIDIA A5000 GPU with batch size 1. Each value is mean $\pm$ standard deviation over 30 timed iterations after 10 warmup iterations. The timed region includes image encoding and action generation, but excludes environment stepping, media writing, and final action unnormalization. Table 8 compares standard Diffusion Policy step budgets with DR schedules using different ReRoll counts.

Sampler	O2/H16/A8	O2/H32/A16	O2/H64/A16
DP, 10 DDIM	10 / 44.7 $\pm$ 1.7	10 / 44.9 $\pm$ 1.6	10 / 46.5 $\pm$ 3.7
DP, 30 DDIM	30 / 121.6 $\pm$ 4.9	30 / 123.6 $\pm$ 2.7	30 / 121.1 $\pm$ 4.4
DP, 50 DDIM	50 / 207.4 $\pm$ 12.6	50 / 203.9 $\pm$ 9.0	50 / 194.8 $\pm$ 3.8
DP, 100 DDPM	100 / 390.7 $\pm$ 7.7	100 / 421.1 $\pm$ 34.6	100 / 387.4 $\pm$ 10.9
DR, 3 ReRoll	20 / 82.8 $\pm$ 5.4	38 / 155.9 $\pm$ 7.3	58 / 229.8 $\pm$ 5.6
DR, 6 ReRoll	29 / 123.1 $\pm$ 2.4	53 / 225.2 $\pm$ 4.7	85 / 346.3 $\pm$ 4.6

Table 8: Policy inference latency on a single RTX A5000 GPU. Each cell reports denoising calls / latency in milliseconds.

B Training Details for Diffusion ReRoll

B.1 Noise Distribution for Training

Unlike Diffusion Forcing, which samples token-wise noise levels independently [14], Diffusion ReRoll trains with structured chunk-wise linear noise profiles. The goal is to expose the model to a broad range of noise levels while preserving the local linear patterns used by the deployment schedule matrix. For the ablations in this section, we mainly use the LIBERO-10 official protocol and AntMaze Giant guidance-based replanning.

B.1.1 Do We Need Linear Chunks for Training?

A model trained with independently sampled token-wise noise levels can in principle be deployed with a ReRoll schedule matrix. However, Fig. 15 shows that this alone is not sufficient. Applying a ReRoll matrix to a Diffusion Forcing model does not reliably improve performance, whereas the DR model with linear-chunk training achieves high performance on both LIBERO-10 and AntMaze Giant.

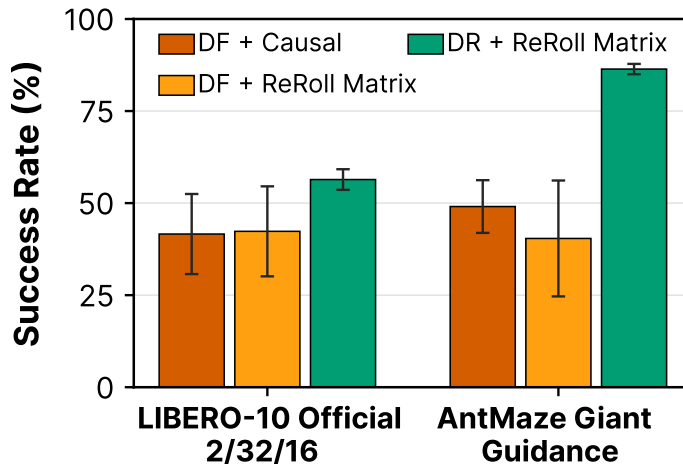


Figure 15: Importance of linear-chunk training. A ReRoll schedule matrix is most effective when the model is trained with structured linear noise chunks.

These results indicate that ReRoll requires alignment between the training noise distribution and the deployment schedule matrix. Linear-chunk training exposes the model to the local piecewise-linear

noise patterns used at inference, providing the structured stochasticity needed for effective chunk-wise revision. This suggests that carefully designed noise patterns can induce useful denoising behavior in generative models.

B.1.2 First-Chunk Noise Assignment

Most chunks are sampled to mimic denoising waves that appear after a ReRoll event. Their starting offsets are therefore sampled from the reset-level range. The first chunk is different because it has no preceding ReRoll event. It must cover both the initial high-noise wave and later deployment states where the beginning of the horizon has already been denoised.

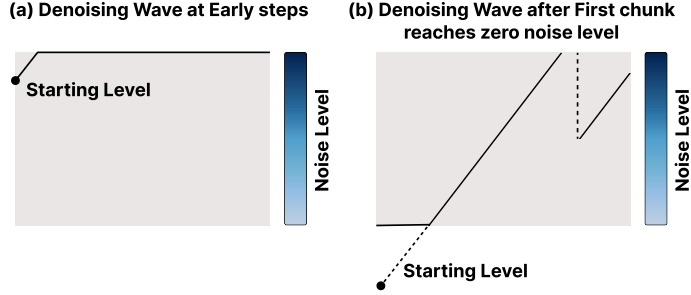


Figure 16: First-chunk noise assignment. Later chunks start from the reset-level range, while the first chunk uses a wider starting-offset range to cover both high-noise initial states and clean-prefix states.

Let K be the maximum noise level and let j denote the local token index within a chunk. Given chunk slope α_c and starting offset s_c , token-wise noise levels are assigned by

$$k_j = \text{clip}_{[0, K]} (\lfloor s_c + \alpha_c j \rfloor). \quad (7)$$

For later chunks, we sample s_c from the reset-level range, e.g., $s_c \sim \mathcal{U}(s_{\min}, s_{\max})$. For the first chunk, we instead use a wider range:

$$s_0 \sim \mathcal{U}(s_{\text{first}}^{\min}, K), \quad s_{\text{first}}^{\min} = \max \left(-K, \left\lfloor -\frac{T}{N_{\text{chunk}}} \alpha_{\min} \right\rfloor \right). \quad (8)$$

where T is the prediction horizon, N_{chunk} is the sampled number of chunks, and $\alpha_{\min}$ is the minimum sampled chunk slope.

Here, s_0 is the starting offset of the first linear profile before clipping. When $s_0 < 0$, clipping produces a clean prefix with noise level 0, matching deployment states where the beginning of the horizon is already denoised. When s_0 is large, the first chunk starts in a high-noise regime, matching the initial denoising wave. In bidirectional training, the same rule is applied symmetrically to the last chunk when it is reversed, since it plays the role of the first denoising wave from the terminal side. This special first-chunk sampling aligns training with the range of first-wave states encountered by the deployment schedule matrix.

B.1.3 Bidirectional Mode Noise Assignment

Bidirectional training uses the same randomized linear-chunk sampler as forward training, but allows selected chunks on the right side of the meeting region to be reversed. For a token with local index j in a chunk of length ℓ_c , a reversed chunk uses

$$j_{\text{rev}} = \ell_c - 1 - j. \quad (9)$$

Thus, the same linear noise rule can represent either a left-to-right or right-to-left denoising wave.

We empirically find that training only with bidirectional chunks is less effective. Instead, we mix forward and bidirectional noise patterns. Each training sample is drawn from one of three branches: (i) forward branch, (ii) masked bidirectional branch, or (iii) full-horizon bidirectional branch.

In the *forward branch*, we use the standard forward training pattern: a random suffix is assigned full noise, and no chunks are reversed. Although bidirectional deployment uses waves from both ends, this branch remains useful because forward local revision still appears in left-origin waves and taking over schedules.

In the *masked bidirectional branch*, we sample a masked interval around the meeting region and assign full noise to tokens inside it. Chunks on the right side of this interval use the reversed coordinate j_{rev} , exposing the model to backward revision from the terminal side.

In the *full-horizon bidirectional branch*, the mask is removed, but a sampled subset of right-side chunks is still reversed. This covers post-meeting deployment states, where the horizon is no longer separated by a masked region.

The masked bidirectional branch is sampled with probability p_{bi} , and the full-horizon bidirectional branch is sampled with probability p_{full} , with

$$p_{\text{bi}} + p_{\text{full}} \leq 1. \quad (10)$$

The remaining probability is assigned to the forward branch.

For tasks with inpainting-style conditioning, we additionally expose the model to near-clean conditioning regions. Conditioned on sampling a bidirectional branch, we apply inpainting-style noise assignment with probability p_{inpaint} . The selected conditioning region is assigned a low noise level sampled from $[0, 0.25K]$. Thus, the overall probability of this case is $(p_{\text{bi}} + p_{\text{full}})p_{\text{inpaint}}$. For example, if $p_{\text{bi}} + p_{\text{full}} = 0.5$ and $p_{\text{inpaint}} = 0.1$, it is applied to 5% of training samples. In maze goal-inpainting, this low-noise region is assigned to the final token, which contains the goal. This has little effect on standard training behavior, but improves the stability of inpainting-style deployment. For the Diffusion Forcing variant used in inpainting, we apply the same masking mechanisms during training while omitting the linear noise-level assignment.

Overall, bidirectional training does not reverse the entire trajectory. It reverses only selected right-side or post-meeting chunks while preserving the randomized linear-chunk distribution used in forward training.

B.1.4 Chunk Slope Range

The chunk-slope range is chosen around the denoising slope used by the deployment ReRoll schedule matrix. If this range is too narrow, the model mainly observes one local linear pattern and can overfit to a specific schedule. We therefore train with a wider slope range so that the model can handle a broader family of linear chunks, including possible schedule variations.

In the main setting, each chunk slope is sampled as

$$\alpha_c \sim \mathcal{U}(0.5s_{\text{dns}}, 2.5s_{\text{dns}}), \quad (11)$$

where s_{dns} is the denoising slope of the deployment schedule. We compare this default wide range with a narrow range,

$$\alpha_c \sim \mathcal{U}(0.875s_{\text{dns}}, 1.125s_{\text{dns}}) \quad (12)$$

Fig. 17 shows that the narrow range slightly degrades performance on both LIBERO-10 and AntMaze Giant. This suggests that ReRoll training benefits from covering diverse local linear noise patterns rather than exactly matching a single deployment schedule.

In our experiments, the deployment schedule uses a fixed denoising slope. However, more flexible schedules could vary the slope over denoising steps or sequence regions. For example, one could use steeper slopes at early denoising steps and shallower slopes later. Training with a wide chunk-slope range may better support such variable-slope schedules.

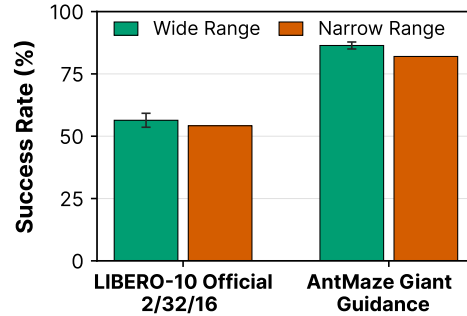


Figure 17: Effect of chunk-slope range. A wider range gives slightly better performance.

B.1.5 Starting-Level Range

The starting-level range controls the initial noise offset of each sampled linear chunk. Although deployment chunks are often introduced near the reset noise level, training only around this narrow range can leave other noise levels underrepresented. For example, if later chunks start only from $[0.25K, 0.75K]$, then noise levels below $0.25K$ are rarely assigned except through the first chunk.

We therefore sample later-chunk starting levels from a zero-to-middle noise range:

$$s_c \sim \mathcal{U}(0, s_{\max}), \quad (13)$$

where $s_{\max}$ is chosen around the middle or reset-level region of the diffusion schedule. This range still covers the reset-level behavior used by the deployment schedule matrix, while also exposing the model to low-noise and nearly clean chunk states.

B.2 Important Components for Diffusion ReRoll Training

Training Diffusion ReRoll requires more than sampling the right token-wise noise levels. Because ReRoll relies on cross-horizon revision, the model also needs compatible representations, masking choices, and architecture interfaces. This section summarizes additional critical components that affect DR performance.

B.2.1 Low Sensitivity to Basic Learning Parameters

Diffusion ReRoll does not require a separate optimization recipe. Across tasks, we largely keep the learning rates, optimizers, batch sizes, and training schedules from the corresponding baseline codebases [14, 29, 4, 30, 25]. Most changes are instead related to token-wise noise assignment, linear-chunk training, masking, and the deployment schedule matrix. When applying DR to new tasks, we therefore recommend first tuning ReRoll-specific schedule and noise-distribution parameters rather than basic learning hyperparameters.

B.2.2 Necessity of Non-causal Training

Many sequence models use causal masking to impose left-to-right sequential structure. However, DR is designed to use cross-horizon context during denoising. Selected regions are re-noised and revised again using information from other parts of the horizon. If causal masking is applied, this information flow is restricted, making it harder for the model to exploit ReRoll’s revisable denoising process.

Fig. 18 shows that adding a causal mask degrades performance on both LIBERO-10 and AntMaze Giant. This supports that DR’s gains depend on non-causal cross-horizon revision, rather than only on token-wise noise conditioning.

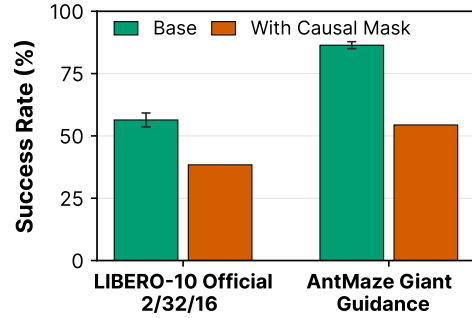


Figure 18: Effect of causal masking. Causal masking restricts cross-horizon information flow and degrades ReRoll performance.

B.2.3 SNR Reweighting

Because ReRoll trains with heterogeneous token-wise noise levels, different tokens can have very different signal-to-noise ratios (SNRs). We do not introduce a new loss weighting rule. Instead, we adopt the Min-SNR weighting strategy [36], following the Diffusion Forcing implementation [14]. For a token at noise level k , we define

$$\text{SNR}(k) = \frac{\bar{\alpha}_k}{1 - \bar{\alpha}_k}. \quad (14)$$

All our experiments at DR and DF use x_0 -prediction. Under this parameterization, we use the clipped SNR weight

$$w_t = \min(\text{SNR}(k_t), \gamma), \quad (15)$$

where k_t is the noise level assigned to token t and γ is the Min-SNR clipping threshold. The resulting weighted objective is

$$\mathcal{L} = \mathbb{E} \left[\sum_{t=1}^H w_t \|x_t^0 - \hat{x}_{\theta,t}^0\|^2 \right]. \quad (16)$$

In our main experiments, we set $\gamma = 2.5$ for both Diffusion Forcing and Diffusion ReRoll, except for the video-generation part of unified video-action generation with larger $\gamma = 5.0$. We found this relatively low clipping threshold to work robustly across tasks, but we did not exhaustively tune SNR weighting. Designing loss weights specifically for structured ReRoll schedule matrices may further improve performance.

B.3 Action Representation for Manipulation

For manipulation tasks, we use an anchor-relative cumulative action representation. Relative action representations are used in some manipulation works, where future end-effector targets are expressed relative to the current end-effector pose rather than a global frame [37, 38]. This representation is useful for Diffusion ReRoll because different horizon tokens may be denoised or re-noised under the schedule matrix. If the model predicts only raw per-step deltas, a late-horizon pose depends on the accumulated accuracy of all previous deltas. As a result, modifying an early token can indirectly change the entire future trajectory. By predicting anchor-relative cumulative targets, each horizon token becomes a more self-contained future pose target, making token-wise revision more stable. We additionally apply horizon-dependent normalization to compensate for the larger scale of later cumulative targets.

Horizon-dependent normalization. Cumulative targets have horizon-dependent scale. Later tokens usually contain larger accumulated motion than earlier tokens. We therefore apply horizon-dependent scaling before diffusion training. For position, the per-step position deltas are first normalized by the standard action normalizer and then accumulated:

$$\tilde{p}_i = \sum_{k=1}^i \bar{\Delta} p_k, \quad (17)$$

where $\bar{\Delta} p_k$ is the normalized position delta. Thus, $\tilde{p}_i$ is a cumulative normalized displacement, not a metric displacement. We scale it as

$$\hat{p}_i = \frac{\tilde{p}_i}{s_i^p}, \quad s_i^p = i^{\alpha_p}. \quad (18)$$

The exponent α_p is estimated from the training-set statistics of cumulative position targets. Specifically, we compute the empirical standard deviation $\sigma_p(i)$ at each horizon index and choose α_p so that the unit-intercept power law matches the geometric trend of $\sigma_p(i)$ across the horizon.

For rotation, we accumulate in physical rotation space rather than normalized action space. Each per-step axis-angle delta is first converted to a rotation, and cumulative rotations are composed on $SO(3)$:

$$\tilde{R}_i = \tilde{R}_{i-1} \exp([\Delta r_i]_{\times}), \quad \tilde{r}_i = \log(\tilde{R}_i), \quad \tilde{R}_0 = I. \quad (19)$$

Here, $[\cdot]_{\times}$ is the skew-symmetric matrix operator, and $\exp$ and $\log$ are the exponential and logarithm maps between the Lie algebra and $SO(3)$. The rotation target is then scaled by

$$\hat{r}_i = \frac{\tilde{r}_i}{s_i^r}, \quad s_i^r = c_r i^{\alpha_r}. \quad (20)$$

The parameters c_r and α_r are fitted from the empirical standard deviation of cumulative rotation targets on the training set. Unlike position, the intercept c_r is used because rotation deltas are accumulated in physical units and are not pre-normalized before composition.

Finally, we apply a per-channel linear normalizer to the scaled cumulative rotation target $\hat{r}_i$, fitted from the training data. The analogous position cumulative normalizer is disabled in our runs because the position branch already accumulates normalized deltas and uses the unit-intercept scale. The gripper command is left in its original per-step form in all reported experiments.

Conversion back to executable actions. At inference, the model outputs normalized cumulative targets. We first undo the horizon-dependent scaling and normalizers, then recover executable per-step actions by differencing consecutive cumulative poses:

$$\Delta p_i = \tilde{p}_i - \tilde{p}_{i-1}, \quad \Delta R_i = \tilde{R}_{i-1}^{-1} \tilde{R}_i, \quad (21)$$

with $\tilde{p}_0 = 0$ and $\tilde{R}_0 = I$. The resulting position delta, rotation delta, and unchanged gripper command form the executable action sequence.

Diffusion Policy with the same representation.

To check whether DR’s gains come mainly from the action representation, we train an additional Diffusion Policy baseline using the same representation used by DF and DR. This ablation separates the effect of the action representation from the effect of ReRoll’s schedule matrix and linear-chunk training.

As shown in Fig. 19, applying the relative cumulative representation to vanilla Diffusion Policy does not improve performance; it substantially reduces success compared with the original delta-action representation. Thus, the gains of DR are not explained by the action representation alone. Since DF and DR use the same action representation, the performance gap between them is more directly attributed to ReRoll’s structured schedule matrix and linear-chunk training.

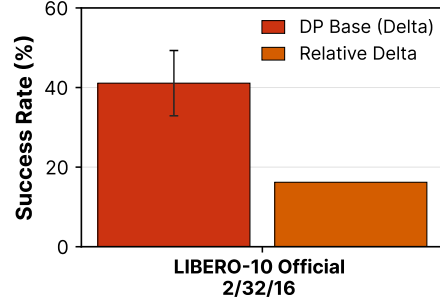


Figure 19: Effect of action representation on Diffusion Policy. The original delta-action representation performs better than the relative cumulative representation used by DF/DR.

B.3.1 Model Architecture

Long-horizon planning. For OGBench maze planning, we use the corresponding architectures from the baseline methods. Diffuser uses its U-Net trajectory denoiser [6], while Diffusion Forcing and Diffusion ReRoll use the transformer sequence denoiser from Diffusion Forcing [14]. Each baseline is trained with its original architecture. Diffusion ReRoll modifies the training noise distribution and deployment schedule matrix, not the backbone architecture.

Policy Learning. For manipulation policy learning, the full-sequence diffusion baseline uses the Diffusion Policy Chi-Transformer [4]. The standard Chi-Transformer conditions on a single global diffusion timestep, inserted into the conditioning stream together with observation tokens. Action tokens use learnable positional embeddings along the prediction horizon.

For Diffusion Forcing and Diffusion ReRoll, we minimally modify this backbone to support per-token noise levels. We remove the global timestep token and inject the noise level directly into each action token. For horizon index i , we compute a sinusoidal embedding $\phi(k_i)$ of the token noise level and concatenate it with the noisy action token $\hat{a}_i$ before projection:

$$z_i = W_2 \sigma(W_1 [\hat{a}_i; \phi(k_i)]). \quad (22)$$

This allows the same denoiser to operate under arbitrary token-wise schedule matrices, including the ReRoll schedule.

We also replace the learnable action-horizon positional embedding with a fixed sinusoidal positional embedding computed from the horizon index. This provides explicit sequence-position information and avoids tying the action-side positional parameters to one fixed horizon length. The observation encoder, observation-side positional embedding, conditioning encoder, transformer decoder,

and output head are kept unchanged. Thus, Diffusion Forcing and ReRoll use the same modified architecture and differ only in the training noise distribution and deployment schedule matrix.

This modification adds only a small number of parameters. For the $H = 16$ policy setting, the Diffusion Policy denoiser contains 8,997,895 parameters, while the per-token-noise-conditioned DF/DR denoiser contains 9,096,199 parameters. The increase is 98,304 parameters, approximately 1.1%, and comes from the action-token input interface.

Unified Video-Action Modeling. For video-action modeling, the full-sequence baseline is UWM [25], which denoises action tokens and future image latents with a DiT-style transformer. The original UWM uses modality-level timestep conditioning through AdaLN: one global timestep is used for the action sequence and one global timestep is used for the future-image sequence. Thus, all action tokens share the same action noise level, and all future-image tokens share the same image noise level at a denoising step.

For UWM + Diffusion Forcing and UWM + ReRoll, we use the same per-token-noise-conditioned architecture. We define a joint action-image horizon and assign a separate noise level to each action token and each future image timestep. The timestep embedding is removed from the AdaLN condition, so AdaLN is conditioned only on the encoded current observation. Instead, each token receives its own sinusoidal noise-level embedding at the input. For action tokens, the noisy action vector is concatenated with the corresponding token-wise noise embedding before the action encoder. For image tokens, the noisy image latent is first patch-embedded; the patch feature is then concatenated with the sinusoidal embedding of the corresponding latent-frame noise level and projected back to the transformer width.

We also replace UWM’s learnable joint positional embedding with fixed sinusoidal positional embeddings over one continuous action-image axis. Image-token positions start after the action-token positions rather than restarting from zero, giving the model an explicit ordering of the joint sequence. Register tokens retain learnable positional parameters because they do not correspond to physical sequence positions.

Finally, we use temporal patch stride one in the image-latent patchifier, so each future latent frame contributes its own patch-token row and can carry its own noise level. All other major components, including the observation encoder, DiT trunk, register tokens, AdaLN observation conditioning, and output decoders, are kept matched to UWM. Therefore, UWM + Diffusion Forcing and UWM + ReRoll differ from the original UWM mainly in per-token noise conditioning, sinusoidal joint positional embeddings, and temporal patching that exposes one noise level per future latent frame.

C Experiment Settings

C.1 Relative gain calculation

The percentage improvements reported in the abstract are relative gains in average success rate, not absolute percentage-point gains. For a baseline average success rate S_{base} and a ReRoll average success rate S_{DR} , we compute

$$\text{Gain} = \frac{S_{\text{DR}} - S_{\text{base}}}{S_{\text{base}}} \times 100.$$

In guidance-based planning, Diffusion Forcing achieves an average success rate of 72.78%, while Diffusion ReRoll achieves 88.07%, giving

$$\frac{88.07 - 72.78}{72.78} \times 100 = 21.0\%.$$

In goal-inpainting planning, Diffuser achieves 73.78%, while Diffusion ReRoll achieves 90.76%, giving

$$\frac{90.76 - 73.78}{73.78} \times 100 = 23.0\%.$$

For LIBERO-10 under the official evaluation protocol, Diffusion Policy achieves an average success rate of 40.06%, while Diffusion ReRoll achieves 62.70%, giving

$$\frac{62.70 - 40.06}{40.06} \times 100 = 56.5\%.$$

Thus, the reported gains summarize relative improvements over the corresponding matched baselines.

C.2 Long-Horizon Planning

C.2.1 Evaluation Settings

We evaluate long-horizon planning on the OGBench PointMaze and AntMaze navigation datasets. The dataset horizon is 500 for Medium and Large mazes and 1000 for Giant mazes. At Diffusion Forcing and Diffusion ReRoll training, we have used 5 frame stack per token for Large and Medium maze and 10 frame stack per token for Giant maze. Total 100 horizon is uniformly used for all the tasks. We mainly follow the MCTD evaluation protocol [29], but re-enable the default OGBench start-goal randomization that is partially removed in MCTD.

We compare Diffuser, Diffusion Forcing, and Diffusion ReRoll under two planning protocols: *guidance-based replanning* and *goal inpainting*. For all methods, the guidance and inpainting results use separately trained models. We train three seeds per configuration, select the best-performing checkpoint for each seed, and report the mean and standard deviation across the three training seeds.

Guidance-based replanning. In the guidance-based protocol, trajectories are sampled with test-time guidance and replanning during execution [31, 32]. For both maze types, we use distance-based guidance that minimizes the distance between planned positions and the goal:

$$\mathcal{G}(x_{1:H}, g) = \sum_{i=1}^H \|x_i - g\|_2^2. \quad (23)$$

Although Diffuser is often evaluated with goal inpainting, we use the same distance-based guidance protocol for all methods. For Diffuser guidance training, only the start state is inpainted, while start and goal state are inpainted for inpainting training. During execution, we use receding-horizon replanning every 50 environment steps. The maximum episode length is 1000 steps for Medium/Large mazes and 1500 steps for Giant mazes.

Goal inpainting. In the inpainting protocol, the start and goal states are fixed in the generated trajectory, and the model predicts the intermediate plan. Unlike guidance-based replanning, the model generates one full-horizon plan at the beginning of the episode and does not replan during execution. The plan horizon is set to the dataset horizon: 500 for Medium/Large and 1000 for Giant. Because the final inpainted token is the goal state, we restrict the execution budget to avoid giving the low-level controller unlimited extra time to solve the task by tracking the goal alone. For the same reason, after the planned trajectory is consumed, the controller tracks the penultimate predicted waypoint rather than the final goal token.

For controlled comparison, Diffusion ReRoll uses a bidirectional schedule matrix in the goal-inpainting setting. We also train an inpainting-compatible Diffusion Forcing variant using the same masking scheme and similar bidirectional schedule matrix as DR, but without ReRoll events. This isolates the effect of re-noising from bidirectional conditioning.

Low-level execution. For PointMaze, we use the heuristic low-level controller from Diffuser [6]. Given current position x_t , velocity v_t , planned position p_t , and planned velocity $\dot{p}_t$, the executed action is

$$a_t = 12.5(p_t - x_t) + 1.2(\dot{p}_t - v_t), \quad (24)$$

clipped to the valid action range. Because OGBench PointMaze does not provide velocity in the original observation, we augment the state with velocity to apply this controller.

For AntMaze, we use the learned low-level controller provided by the MCTD codebase [39, 29]. The high-level diffusion planner outputs desired positions. During execution, the low-level policy tracks a subgoal 10 planned steps ahead. When the ant comes within distance 1.0 in the raw AntMaze (x, y) coordinate space of the current subgoal, the subgoal is advanced by another 10 planned steps. If the planned trajectory is exhausted, the controller tracks the final available planned position.

C.2.2 ReRoll Parameters for Long-Horizon Planning

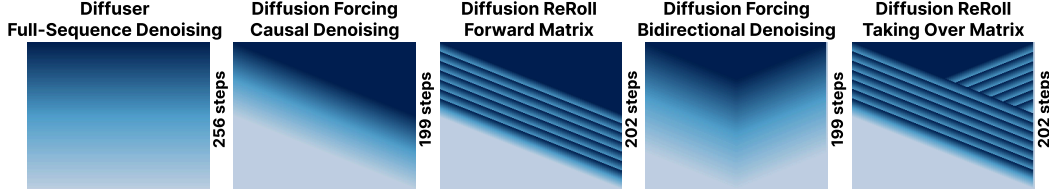


Figure 20: Used schedule matrices at OGBench planning tasks.

For maze planning, we use a prediction horizon of $T = 100$ for the ReRoll schedule matrix. The deployment matrix uses denoising slope $s_{\text{dns}} = 4$ and 6 ReRoll events. The schedule matrices are shown in Fig. 20. This setting creates denoising waves that move across the horizon with moderate temporal span, while allowing repeated re-noising over most of the planned trajectory. Specific parameters are shown in Table 9.

Parameter	Value
Prediction horizon T	100
Denoising slope s_{dns}	4
ReRoll events N_{roll}	6
Reset noise level k_{reset}	$0.52K$
Training chunk count N_{chunk}	1–8
Chunk slope range	[2, 10]
Starting-level range	$[0, 0.5K]$
Forward branch probability p_{forward}	0.475
Bidirectional branch probability p_{bi}	0.475
Full-horizon branch probability p_{full}	0.05
Inpainting-mask probability p_{inpaint}	0.1

Table 9: ReRoll hyperparameters for maze planning.

C.2.3 Training and Sampling Hyperparameters

Table 10 summarizes the main training and sampling hyperparameters for the maze-planning experiments. Diffuser follows the full-sequence diffusion setup, while Diffusion Forcing and Diffusion ReRoll share the same transformer backbone and differ only in the training noise distribution and deployment schedule matrix. For each training seed, we evaluate 7 checkpoints and select the best-performing checkpoint.

C.3 Policy Learning

C.3.1 Evaluation Environment Settings

We evaluate diffusion-policy-style action prediction on LIBERO-10 multi-task learning and RoboCasa single-task learning. LIBERO-10 contains ten long-horizon manipulation tasks using a shared 7-DoF Franka end-effector action space. Among various RoboCasa benchmark tasks, we have chosen four single-task kitchen manipulation settings: CloseSingleDoor, CoffeePressButton, CoffeeServeMug, and PnPCounterToSink. RoboCasa actions are 12-dimensional, including arm motion, gripper command, base motion, and control mode. However, for the selected tasks, only seven action dimensions are used, so we train with the same 7-D action format as LIBERO-10.

Hyperparameter	Diffuser	DF / DR
Optimizer	AdamW	AdamW
(β_1, β_2)	(0.9, 0.999)	(0.9, 0.999)
Learning rate	2×10^{-4}	5×10^{-4}
Weight decay	10^{-4}	10^{-4}
Warmup steps	10,000	10,000
LR schedule after warmup	constant	constant
Batch size	32	2048
Max training steps	1.2M	130,005
Evaluated checkpoints	500k–1.1M, every 100k	70k–130k, every 10k
EMA	decay 0.995	not used
EMA start / update	2000 / every 10 steps	–
Training diffusion steps	256	1000
Noise clip	20.0	20.0
Beta schedule	cosine	linear
Prediction objective	x_0 prediction	x_0 prediction
Loss weighting	uniform elementwise MSE (no SNR weighting)	min-SNR, clip 2.5
Guidance scale	0.1	2.0
Training window	task horizon	100 tokens
Backbone	Temporal U-Net	Transformer
Backbone size	channels [32, 128, 256], kernel 5	width 128, 12 layers, 4 heads, FFN 512

Table 10: Training and sampling hyperparameters for OGBench PointMaze and AntMaze planning.

Observations. LIBERO-10 uses two RGB cameras together with proprioceptive state consisting of end-effector position, end-effector quaternion, and gripper state. We additionally provide a scalar `task_uid` as the only task identifier. No language instruction or pretrained language encoder is used. RoboCasa uses three cameras with the same proprioceptive channels except for the task identifier.

Encoders and data splits. For LIBERO-10, all policies are trained from scratch. The image streams are encoded by a Robomimic-style convolutional visual encoder with random crop augmentation [40], and no vision encoder or language pretraining is used. Each LIBERO task provides 50 demonstrations. For each training seed, we split them into 45 training and 5 validation trajectories per task, yielding 450 training trajectories in total. The split is regenerated across the three training seeds.

For RoboCasa, fully from-scratch visual encoders did not reliably converge at the demonstration counts used in our experiments. We therefore initialize the per-camera image encoder with the standard ImageNet-pretrained ResNet-18. This is the only pretrained component in the RoboCasa setup. CloseSingleDoor, CoffeePressButton, and CoffeeServeMug use 50 demonstrations each for training, 4 demonstrations for validation. PnPCounterToSink uses 300 demonstrations for training, including 50 human demonstrations and 250 MimicGen-generated demonstrations, with 24 demonstrations used for validation [41].

Both benchmarks process 128×128 camera inputs through the Robomimic image encoder, but use different random-crop sizes consistent with their encoder initialization. LIBERO-10 uses an aggressive 76×76 random crop with from-scratch weights, whereas RoboCasa uses a milder 116×116 random crop together with an ImageNet-pretrained ResNet-18 backbone [40]. The crop size and pretraining choice are therefore coupled and are held fixed across all three compared methods within each benchmark.

Checkpoint Selection For policy-learning experiments, we select the best checkpoint for each training seed using validation rollouts. After selecting the best checkpoint for each seed, we report test performance as the mean and standard deviation across three training seeds.

For LIBERO-10, we evaluate late-stage checkpoints every 10 epochs. All three methods are evaluated at 25 checkpoints from 210 epochs to 450. Each candidate checkpoint is evaluated with 50 initializations per task, and the checkpoint with the highest success rate is selected.

For RoboCasa, checkpoint selection is performed similarly. For 50-demonstration single-task experiments, we evaluate 31 candidate checkpoints from epochs 500 to 2000 at intervals of 50 epochs. For the 300-demonstration Pick-and-Place Counter-to-Sink setting, we evaluate 15 candidate check-

points from epochs 220 to 500 at intervals of 20 epochs. Each candidate checkpoint is evaluated with 50 validation rollouts.

C.3.2 LIBERO-10 Reset Protocol

In the main experiments, we report LIBERO-10 results under two reset protocols: the official initialization and an open-gripper initialization. This distinction is important because the initial gripper state has a large effect on evaluation difficulty. In the training demonstrations, the robot usually starts with an opened gripper. However, under the official evaluation reset, the robot waits after object spawning without actively opening the gripper, which produces an initial gripper state farther from the training distribution. The open-gripper protocol instead opens the gripper during this waiting period, making the evaluation state closer to the demonstrations.

Fig. 21 shows this distribution gap. The average initial gripper value is 0.07246 m in the training data and 0.06811 m under the open-gripper reset, but only 0.04135 m under the official reset. Thus, the official reset creates a stronger train-test mismatch in the initial robot state.

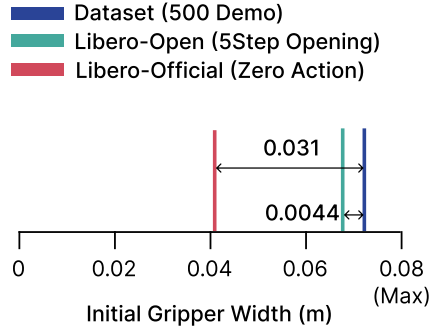


Figure 21: Initial gripper-state gap in LIBERO-10. The open-gripper reset is closer to the training distribution, while the official reset produces a more closed initial gripper state.

This protocol difference raises two concerns. First, reported LIBERO-10 results are difficult to compare unless the reset behavior is specified. LIBERO-10 is widely used in VLA and manipulation-policy benchmarks, but some comparisons reuse success rates from prior papers rather than re-running all methods under a unified evaluation script. In inspecting public codebases, we found that some implementations use different gripper-reset behavior during object spawning, which can unintentionally mix evaluation protocols across reported results. We do not claim that all discrepancies in prior results come from this issue, since we did not re-evaluate prior works under both protocols. However, our results show that this small implementation detail can noticeably affect success rates, so LIBERO-10 evaluations should explicitly report the initialization procedure.

Second, the open-gripper reset can make LIBERO-10 substantially easier. For example, in the 2/16/8 setting, the standard Diffusion Policy baseline reaches 76.2% success under the open-gripper reset but only 28.1% under the official reset. This gap suggests that a major part of the task difficulty comes from recovering from an initial gripper state that is out of distribution relative to the demonstrations. Therefore, we report both protocols in the main paper, the official reset for benchmark fidelity, and the open-gripper reset to show performance under an initialization closer to the training distribution.

C.3.3 Applied ReRoll Parameters

We use nearly shared ReRoll parameters across policy-learning experiments to avoid per-task schedule search. For the main evaluation, 100 Denoising Diffusion Probabilistic Model (DDPM) steps are used for basic diffusion policy. Since policies are executed in a receding-horizon manner, the entire predicted horizon does not need to be fully resolved at every inference call. The default parameters are summarized in Table 11.

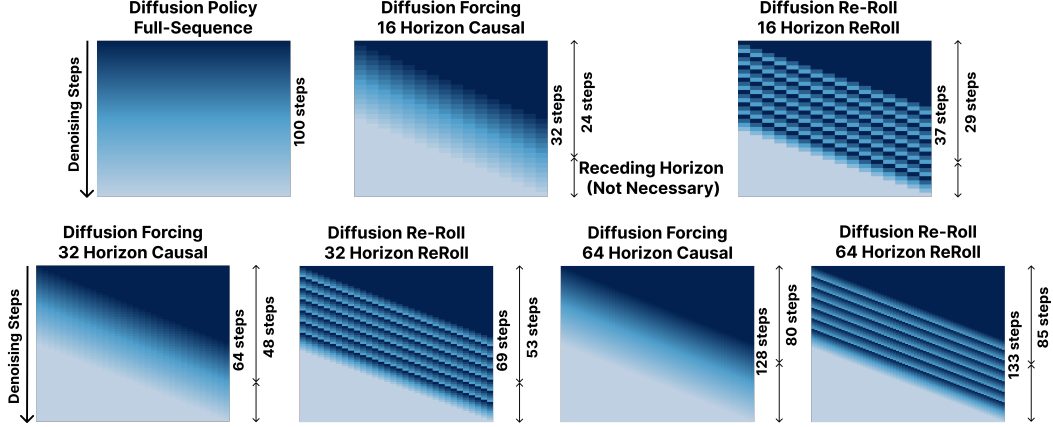


Figure 22: Schedule matrices used for diffusion-policy-style action prediction. Under receding-horizon execution, only the executable prefix of the predicted action chunk is applied before the next inference call.

Parameter	Value
Prediction horizon T	16/32/64
Denoising slope s_{dns}	4
ReRoll events N_{roll}	6
Reset noise level k_{reset}	$0.5K$
Training chunk count N_{chunk}	1–8
Chunk slope range	$[2, 10]$
Starting-level range	$[0, 0.5K]$

Table 11: ReRoll hyperparameters for policy learning.

C.3.4 Basic Learning Hyperparameters

All three policy-learning methods use the same Chi-Transformer backbone and the same optimizer settings. Table 12 summarizes the main hyperparameters.

The basic learning hyperparameters, including optimizer settings, learning rates, and batch size are inherited from the OAT codebase [30]. The final reported test success is averaged over three training seeds.

C.4 Unified Video-Action Modeling

C.4.1 Evaluation Environment Settings

For the unified video-action world modeling experiments, we evaluate UWM, UWM + Diffusion Forcing (DF), and UWM + Diffusion ReRoll (DR) on LIBERO-10. Image observations are encoded by a from-scratch ResNet visual backbone. All three models are pretrained on LIBERO-90 and fine-tuned separately on each LIBERO-10 task. For OOD evaluation, we enlarge the object initialization range by 0.03 m and hide task-irrelevant background objects.

Joint horizon and action-execution protocol.

The original UWM uses a 16-action chunk with 2 future observation latents. We instead use 32 action tokens with the same 2 future image latents for all three variants, giving the joint sequence



Figure 23: Observation differences between the base and OOD settings. Task-irrelevant background objects are removed in the OOD setting.

Hyperparameter	Diffusion Policy	DF / DR
Backbone	Chi-Transformer	Chi-Transformer
Action positional embedding	learnable	sinusoidal
Noise conditioning	global timestep token	per-token noise embedding
Causal attention mask	used	not used
Dropout	0.1	0.01
Optimizer	AdamW	AdamW
(β_1, β_2)	(0.9, 0.95)	(0.9, 0.95)
Policy learning rate	5×10^{-5}	5×10^{-5}
Visual encoder learning rate	1×10^{-5}	1×10^{-5}
Weight decay	0	0
Warmup steps	200	200
LR schedule after warmup	constant	constant
Batch size	256	256
EMA	power EMA	power EMA
Training epochs	451	451
Training diffusion steps	100	100
Inference steps	100 DDPM	schedule-matrix DDIM
Prediction target	ϵ	x_0
Beta schedule	squared cosine	squared cosine
Loss weighting	none	min-SNR, clip 2.5

Table 12: Training and inference hyperparameters for policy-learning experiments. DF denotes Diffusion Forcing and DR denotes Diffusion ReRoll.

layout

$$T_{\text{joint}} = \underbrace{T_a}_{32 \text{ action tokens}} + \underbrace{T_v}_{2 \text{ future image latents}} = 34. \quad (25)$$

For policy evaluation, every predicted action is executed before the next inference call, corresponding to the 2/32/32 configuration: 2 observation frames, 32 predicted actions, and 32 executed actions. We evaluate two policy-generation modes. In *action-only generation*, only the action tokens are generated, while the future image tokens are kept at full noise. In *joint generation*, action tokens and future image tokens are denoised together under the joint schedule matrix. After environment reset, the policy is queried immediately from the post-spawn observation without additional gripper-opening or object-settling actions, roughly consistent with the official LIBERO-style reset described in Appendix C.3.2.

Inverse dynamics setting. For inverse-dynamics evaluation, the model receives the current observation o_t and a future observation o_{t+T_a} from the demonstration, and predicts the action chunk connecting them. We implement this by inpainting the future image latents in the joint schedule matrix, so the future visual tokens are fixed as clean end-conditions while the 32 action tokens are denoised. Execution follows the demonstration timeline. After each inference call, all 32 predicted actions are executed and the demonstration index is advanced by 32. The episode ends when the demonstration is consumed, the LIBERO success flag is reached, or the rollout-length budget is exhausted. No additional terminal retry time is given, so inverse-dynamics success is evaluated under a stricter time budget than policy evaluation.

C.4.2 Difference Between Action and Image Generation

In the UWM + DF and UWM + DR settings, action tokens and image-latent tokens are denoised within the same joint sequence. We use the x_0 -prediction protocol and generate images with fewer DDIM steps than the original UWM sampler. Under this shorter schedule, using the same loss weighting for action and image tokens did not train the image-generation part sufficiently well. Since action targets and image-latent targets have different scales and noise-level behavior, we apply a small modality-specific adjustment only for the DF/DR variants.

Asymmetric min-SNR clipping. We use min-SNR loss weighting [36] for both action and image losses, but with different clipping thresholds. The action loss uses $\gamma_{\text{act}} = 2.5$, while the image-latent loss uses $\gamma_{\text{img}} = 5.0$. This gives the image branch a broader effective SNR range during training.

Image-loss weight. We also increase the image dynamics loss weight:

$$\mathcal{L} = \mathcal{L}_{\text{action}} + w_{\text{img}}\mathcal{L}_{\text{image}}, \quad w_{\text{img}} = 1.5. \quad (26)$$

This adjustment improves joint video-action generation by preventing the image-latent objective from being underweighted relative to the action objective. All other diffusion-side settings, including the prediction target, beta schedule, sample clipping, and train/inference step counts, are kept the same for the two streams.

C.4.3 Applied ReRoll Parameters

For UWM + DR, we use the bidirectional *meeting* schedule matrix. The schedule is defined over the joint action-image horizon with $T_a = 32$ action tokens and $T_v = 2$ future image-latent tokens, giving $T_{\text{joint}} = 34$. Nearly the same schedule parameters are used for marginal action-policy generation, joint action-image generation, and inverse dynamics. In marginal action-policy generation, the image-latent columns are kept at full noise, so only the 32 action tokens are used for control. In inverse dynamics, the future image-latent columns are fixed as clean inpainting constraints, so the free denoising variables are again the 32 action tokens.

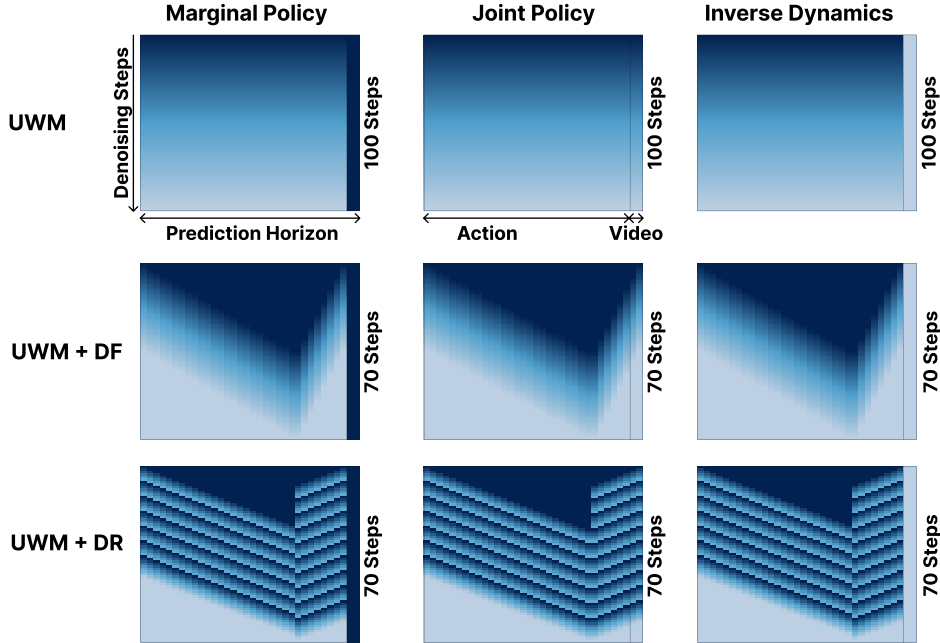


Figure 24: Unified Video-Action generation schedule matrices. The schedule is defined over 32 action tokens and 2 future image-latent tokens. For inverse dynamics, the future image columns are clamped as clean inpainting constraints.

C.4.4 Basic Learning Hyperparameters

All three methods use the same DiT-style joint transformer backbone and optimizer settings whenever possible. UWM + DF and UWM + DR differ from UWM mainly in the per-token noise-conditioning interface. Parameters are shown at Table 14.

Parameter	Value
Joint horizon T_{joint}	34
Action horizon T_a	32
Meeting point	$0.765T_{\text{joint}} / 0.75T_a$
Denoising slope s_{dns}	$3.78(T_{\text{joint}}) / 3.56(T_a)$
ReRoll events, forward N_{roll}	6
ReRoll events, backward N_{roll}	9
Reset noise level k_{reset}	$0.412K$
Training chunk count N_{chunk}	1–8
Chunk slope range	$[2, 10]$
Starting-level range	$[0, 0.5K]$
Bidirectional branch probability p_{bi}	0.30
Full-horizon branch probability p_{full}	0.05
Image-inpainting probability p_{inpaint}	0.1

Table 13: ReRoll hyperparameters for unified video-action modeling. For entries with two values, the first corresponds to the joint action-image schedule and the second to the action-only schedule.

Hyperparameter	UWM	UWM + DF / DR
Backbone	DiT-style joint transformer	DiT-style joint transformer
Embedding dimension	768	768
Depth / heads / MLP ratio	12 / 12 / 4	12 / 12 / 4
Register tokens	24	24
Image latent patch shape	$2 \times 4 \times 4$	$1 \times 4 \times 4$
Positional embedding	learnable	sinusoidal
Noise conditioning	global AdaLN	per-token input concat.
Optimizer	AdamW	AdamW
$(\beta_1, \beta_2), \epsilon$	$(0.9, 0.999), 10^{-8}$	$(0.9, 0.999), 10^{-8}$
Learning rate, pretrain	1×10^{-4}	1×10^{-4}
Learning rate, finetune	1×10^{-4} , cosine, 1k warmup	1×10^{-4} , cosine, 1k warmup
Weight decay	10^{-6}	10^{-6}
Pretraining steps	100,000	100,000
Fine-tuning steps	20,000	20,000
Batch size, train / eval	16 / 4	16 / 4
Observation history	2	2
Visual backbone	ResNet	ResNet
Image resize / crop	$240 \times 240 / 224 \times 224$	$240 \times 240 / 224 \times 224$

Table 14: Training and inference hyperparameters for unified video-action modeling.

D Additional Results

D.1 Best-Checkpoint Results

The main results report mean performance over three training seeds after selecting the best checkpoint for each seed. For completeness, we also report the best single-seed performance observed for each method and setting. These results are not used as the primary comparison, but show the maximum performance reached by each method under the evaluated checkpoint set. Table 15 reports best-checkpoint results for long-horizon planning, while Tables 16 and 17 report best-checkpoint results for LIBERO-10 and RoboCasa policy learning.

Mode	Method	AntMaze			PointMaze		
		Giant	Large	Medium	Giant	Large	Medium
Guidance Replan	Diffuser	17.2	46.8	70.0	32.0	31.6	47.2
	Forcing	<u>56.0</u>	<u>61.6</u>	<u>96.0</u>	<u>92.8</u>	<u>62.0</u>	<u>94.0</u>
	ReRoll	87.6	71.2	98.4	100.0	78.4	99.2
Inpainting	Diffuser	34.4	<u>72.8</u>	89.2	66.8	96.4	<u>95.2</u>
	Forcing	<u>75.6</u>	68.0	<u>91.2</u>	<u>96.4</u>	<u>97.6</u>	94.4
	ReRoll	84.8	74.8	94.4	99.2	100.0	99.2

Table 15: Maximum long-horizon planning success rates (%) on OGBench PointMaze and AntMaze. Values report the best observed success rate for each method and task. Best results are bolded and second-best results are underlined.

Method	LIBERO-10 Official (Obs./Hor./Act.)					LIBERO-10 Open (Obs./Hor./Act.)				
	2/16/8	2/32/16	2/64/16	4/32/16	8/32/16	2/16/8	2/32/16	2/64/16	4/32/16	8/32/16
Diff. Policy	31.6	49.6	38.2	52.2	54.7	<u>76.6</u>	<u>67.6</u>	60.2	70.6	66.0
Forcing	<u>47.2</u>	<u>50.8</u>	<u>46.0</u>	<u>65.4</u>	<u>65.0</u>	69.4	61.4	<u>62.0</u>	<u>71.6</u>	<u>77.8</u>
ReRoll	68.8	59.4	67.8	76.6	73.8	85.0	77.8	71.8	83.8	83.8

Table 16: Maximum LIBERO-10 multi-task policy learning success rates under two initialization protocols. Values report the best observed success rate (%) for each method and setting. Best results are bolded and second-best results are underlined.

Method	CloseSingleDoor			CoffeePressButton			CoffeeServeMug			PnPCounterToSink (300 Demos)		
	2/32/16	2/64/16	4/32/16	2/32/16	2/64/16	4/32/16	2/32/16	2/64/16	4/32/16	2/32/16	2/64/16	4/32/16
Diff. Policy	54.0	54.0	66.0	82.0	64.0	84.0	46.0	36.0	46.0	34.0	26.0	26.0
Forcing	<u>66.0</u>	70.0	<u>68.0</u>	<u>88.0</u>	<u>72.0</u>	94.0	42.0	46.0	<u>40.0</u>	34.0	<u>30.0</u>	38.0
ReRoll	74.0	<u>66.0</u>	70.0	90.0	88.0	<u>90.0</u>	46.0	<u>40.0</u>	38.0	32.0	36.0	<u>32.0</u>

Table 17: Maximum RoboCasa single-task policy learning success rates (%). Values report the best observed success rate for each method and setting. Best results are bolded and second-best results are underlined.

D.2 Details for the Zero-Action Intervention

In the main paper, we report zero-action error to evaluate action-video consistency. This diagnostic tests whether the model can generate future visual dynamics consistent with an imposed zero-action condition, and whether inverse dynamics recovers actions consistent with the generated video.

In the zero-action intervention, we fix the future action tokens to zero and generate future image tokens with the forward-dynamics mode. We then apply inverse dynamics to the generated future images and measure the terminal end-effector displacement implied by the recovered actions. Lower displacement indicates stronger consistency with the imposed zero-action condition.

For reference, successful demonstration trajectories in the dataset have an average terminal end-effector displacement of 0.351 m over the same horizon. We also report the reduction from this demonstration displacement,

$$\Delta_{\text{demo}} = d_{\text{demo}} - d, \quad (27)$$

where $d_{\text{demo}} = 0.351$ m and d is the measured zero-action displacement. A larger Δ_{demo} means the generated behavior implies less motion than a normal successful demonstration, as desired under the zero-action condition.

Evaluation setting	Dataset ref.	UWM	UWM + DF	UWM + DR
Successful demonstrations	0.351 ($\Delta = 0.000$)	–	–	–
Gen. zero-act video $\rightarrow$ Inv. Dyn.	–	0.324 ($\Delta = 0.027$)	0.331 ($\Delta = 0.020$)	0.301 ($\Delta = 0.050$)
GT zero-act video $\rightarrow$ Inv. Dyn.	–	0.310 ($\Delta = 0.041$)	0.332 ($\Delta = 0.019$)	0.295 ($\Delta = 0.056$)

Table 18: Zero-action consistency diagnostics. Each cell reports terminal end-effector displacement d in meters, with $\Delta_{\text{demo}} = 0.351 - d$ in parentheses. Lower d and larger positive Δ_{demo} indicate stronger consistency with the imposed zero-action condition.

All models produce smaller implied displacements than the successful-demonstration reference, showing that they respond to the zero-action condition. In the generated-video setting, UWM + DR produces the lowest displacement and the largest reduction from demonstration motion.

The GT zero-action video setting further isolates the inverse-dynamics behavior. Because the future images are provided from the true zero-action rollout, this setting removes errors from video generation and should make the desired zero-motion action easier to infer. For UWM and UWM + DR, the displacement decreases when using GT zero-action video compared with generated zero-action video. In contrast, UWM + DF does not improve and shows slightly larger displacement, suggesting that its inverse-dynamics prediction is less well aligned with the given zero-motion visual transition. Since UWM + DF and UWM + DR share the same architecture, this suggests that DR’s structured training noise distribution and revisable denoising schedule help the model align video-conditioned action prediction with the imposed zero-action dynamics.

D.3 High Variance in LIBERO-10 Official 2/16/8

In Table 2 of the main paper, DR shows high variance in the LIBERO-10 Official 2/16/8 setting. We do not interpret this as a general instability of ReRoll. As discussed in Appendix C.3.2, the official reset creates an out-of-distribution initial gripper state relative to the demonstrations. The short-horizon 2/16/8 setting appears especially sensitive to this mismatch. If the policy fails to recover from the initial gripper state, official success can drop sharply.

Table 19 shows the per-seed results under both the official and open-gripper reset protocols. For each seed, DP, DF, and DR use the same 10% validation split. Under the official reset, all three methods show at least one weak seed, and DR has one particularly low seed. Under the open-gripper reset, where the initial state is closer to the training distribution, all DR seeds perform strongly and consistently. This suggests that the large official-setting deviation is mainly driven by the OOD reset condition and seed-dependent data splits, rather than by short-horizon ReRoll behavior alone. This effect may also be amplified because our LIBERO-10 policies are trained from scratch without pretrained vision encoders or text encoders, making recovery from the OOD initial gripper state more difficult.

Seed	Official			Open		
	DP	DF	DR	DP	DF	DR
17	28.6	24.0	23.6	76.4	66.4	78.4
23	31.6	40.8	60.6	76.6	69.4	85.0
42	24.0	47.2	68.8	75.6	69.2	84.2
Mean	28.1 ± 3.8	37.3 ± 12.0	51.0 ± 24.1	76.2 ± 0.5	68.3 ± 1.7	82.5 ± 3.6

Table 19: Per-seed LIBERO-10 2/16/8 success rates under official and open-gripper resets. The open-gripper reset reduces the initial-state mismatch and produces more stable performance.

D.4 Long-Horizon Planning Visualization

Fig. 8 shows qualitative denoising behavior on AntMaze Giant. We provide additional visualizations for Diffuser, Diffusion Forcing, and Diffusion ReRoll in Figs. 25–27. These examples illustrate different failure and recovery patterns: full-sequence diffusion can commit to early denoising errors, causal denoising can lock in incorrect prefixes, while Diffusion ReRoll keeps plans revisable through selective re-noising.

D.5 Generated Video Visualization

We provide generated-video examples in Figs. 28 and 29. UWM, UWM + DF, and UWM + DR produce comparable visual generation quality in these examples. The original UWM uses 100 DDIM steps, while UWM + DF and UWM + DR use 70 DDIM steps. These results suggest that the downstream performance differences are not caused by a clear degradation in generated-video quality.



Figure 25: Denoising visualization for Diffuser. We visualize predicted clean trajectories $\hat{x}_0$ at selected denoising steps. Full-sequence diffusion can commit to an early incorrect plan.



Figure 26: Denoising visualization for Diffusion Forcing. We visualize predicted clean trajectories $\hat{x}_0$ at selected denoising steps. Causal denoising can resolve early prefixes before later horizon regions are sufficiently revised.

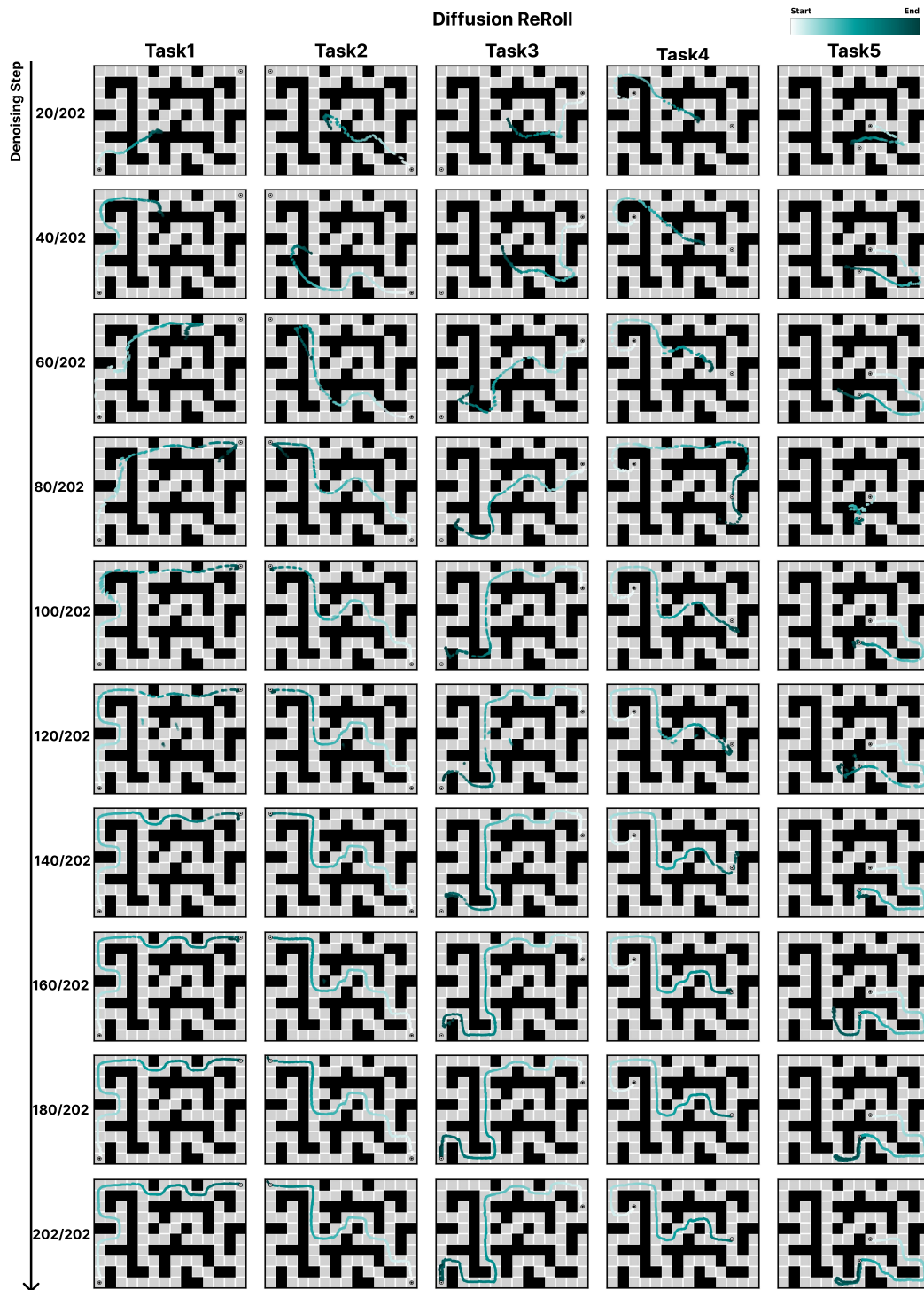


Figure 27: Denoising visualization for Diffusion ReRoll. We visualize predicted clean trajectories $\hat{x}_0$ at selected denoising steps. ReRoll keeps trajectory regions revisable and can recover a valid long-horizon plan.

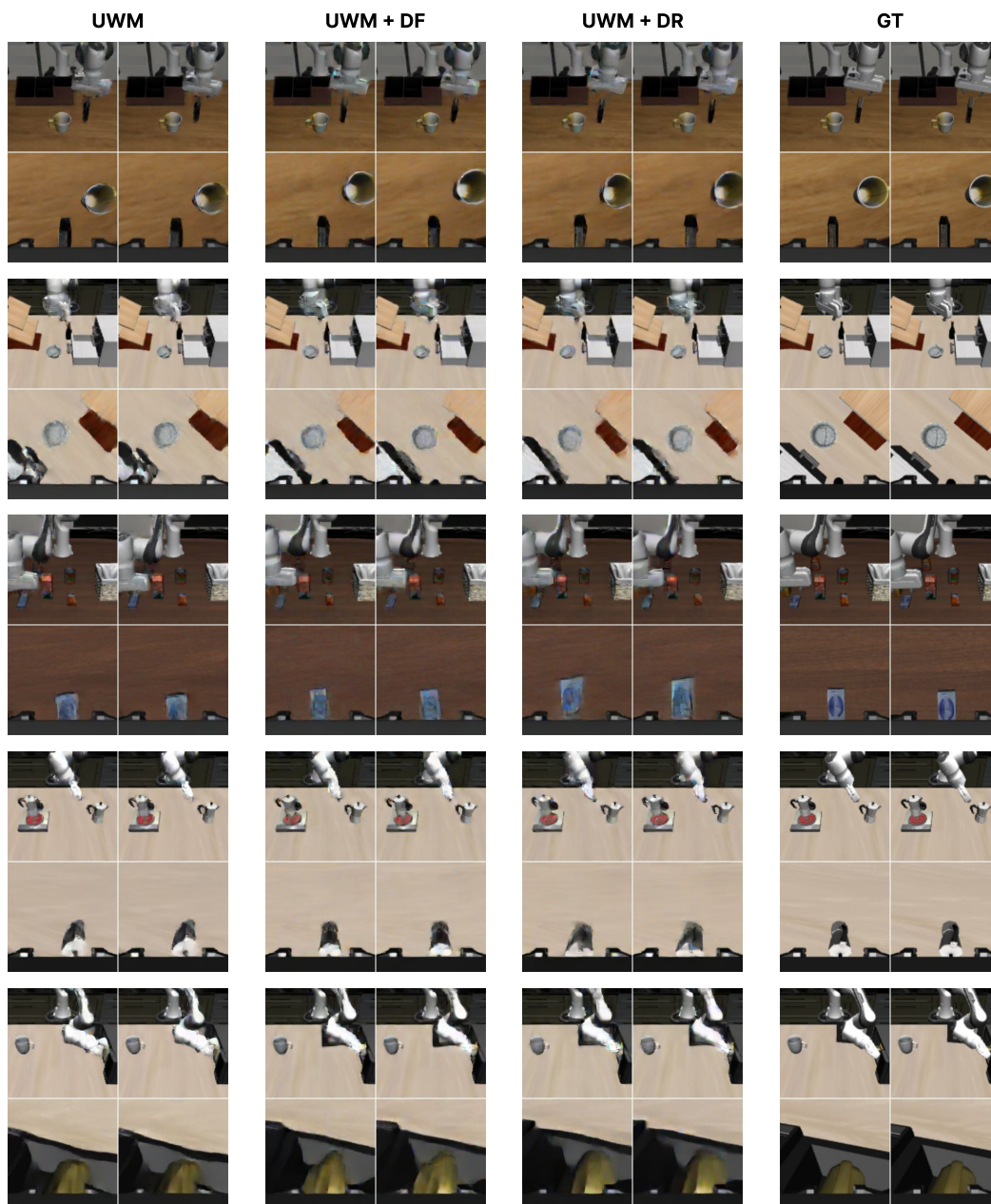


Figure 28: Generated future-video examples from UWM, UWM + DF, and UWM + DR.

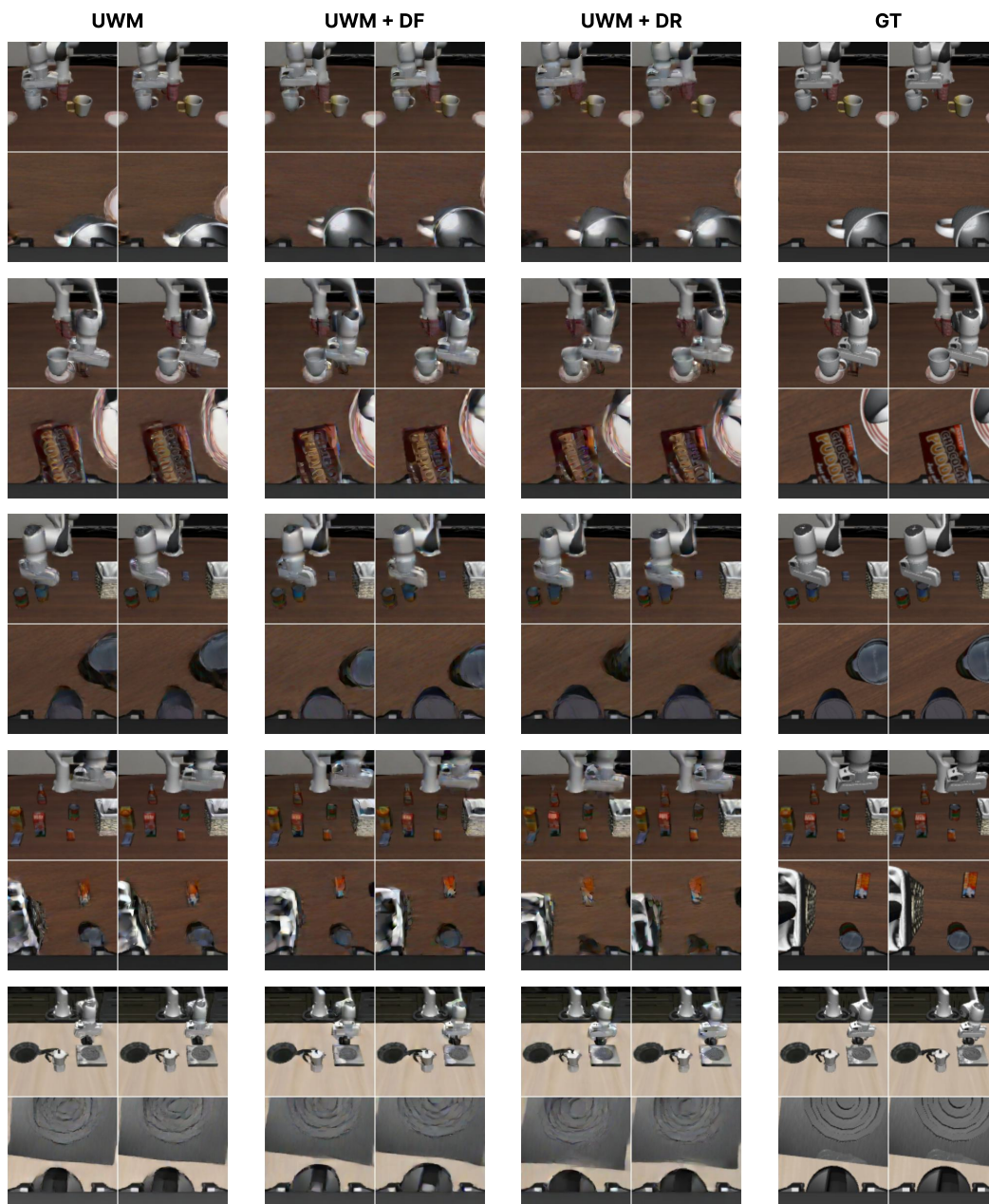


Figure 29: Additional generated future-video examples from UWM, UWM + DF, and UWM + DR.