RECURSIVE SUCCESSOR MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Planning with world models enables improvement beyond the dataset, but compounding prediction errors hinder long-horizon planning. To prevent one-step compounding error, we instead model longer-term future state distributions, known as successor measures. We introduce *Recursive Successor Models*, which combine (1) a stable supervised training objective that avoids bootstrapping with (2) a recursive sampling procedure for planning that composes short-horizon distributions into long-horizon distributions. We demonstrate the capabilities of RSM in simulated benchmarks and real-world robot experiments by showcasing improvements in policy performance, planning latency, and training convergence speed across long-horizon and visual tasks.

1 INTRODUCTION

World models (Ha & Schmidhuber, 2018) learn environment dynamics by predicting the future state conditioned on the current state and action. These predictions can in turn be used for sequential decision-making via planning (Hansen et al., 2022; Zhou et al., 2025) or policy improvement (Hafner et al., 2020; Li et al., 2025). A major appeal of this paradigm is that dynamics can be learned with supervised learning objectives directly from offline trajectories of arbitrary quality. However, using such models to predict long-horizon outcomes typically requires repeatedly applying the learned dynamics model. Errors therefore accumulate with planning horizon, limiting the reliability of long-horizon predictions (Janner et al., 2019).

To make decisions, agents need to estimate the long-term value of their actions, often denoted by a Q function in reinforcement learning. A key insight underlying our approach is that estimating the long-term value of an agent's actions does not require predicting precisely *when* certain futures occur, but rather how *how likely* they are to occur. In particular, the value of an action depends on the distribution of future states it induces, rather than the particular sequence of intermediate states used to reach them. This motivates us to learn a model of the policy-dependent long-term future state *distribution*, rather than a standard world model of immediate next state dynamics. Planning by sampling from this multi-timescale distribution to estimate value reduces the number of sequential model predictions required for planning, mitigating compounding error over long horizons.

This distribution of future visitation is a well-studied object in reinforcement learning known as the *Successor Measure* (SM; Dayan 1993). While traditionally difficult to represent, recent advances in generative models have led to practical implementations of SMs (Janner et al., 2020; Farebrother et al., 2025). These methods commonly exploit the Bellman recursion of the successor measure and train with Temporal Difference (TD; Sutton 1988) objectives that bootstrap from the model's own predictions. TD learning enables composing different behaviors within the dataset but is notoriously unstable to train with growth of the horizon and state-space (van Hasselt et al., 2018). In this work, we ask whether successor models can retain a simple supervised training objective while recovering long-horizon composition only when it is needed at inference time.

We present *Recursive Successor Models* (RSM), a method for training and planning with successor models. RSM separates successor learning from long-horizon composition: the model is trained with a simple Monte-Carlo generative objective on observed future states, without bootstrapping, while a recursive sampler composes these predictions into longer-horizon successor distributions at test time. This allows RSM to have a simple, scalable training objective, while retaining the compositional benefits of TD learning.

Our main contributions are: 1) a scalable recipe for learning a successor model from offline datasets that enables long-horizon model-based planning 2) empirical demonstrations that this approach im-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

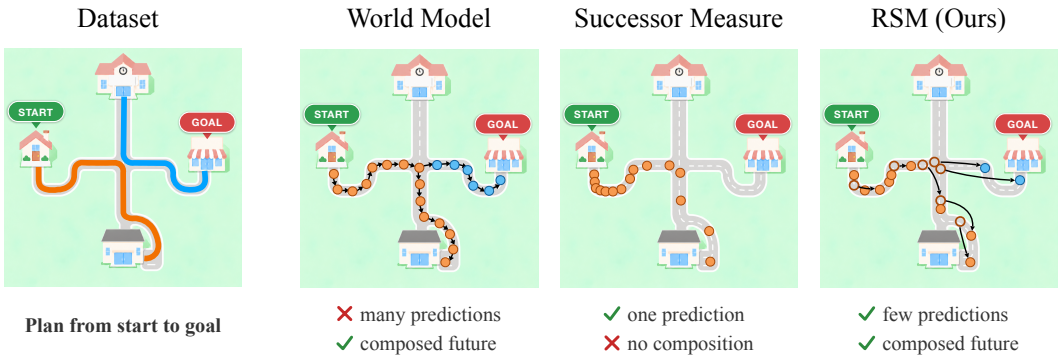


Figure 1: **Paradigms of model-based planning.** Planning by autoregressively rolling out one-step models can lead to compounding errors. Successor measures trained with Monte-Carlo sampling avoid compounding errors but only generate futures from the same trajectory without composition. RSM uses a few "hops" of short-horizon Monte-Carlo models to estimate a composed long-horizon successor distribution, requiring fewer predictions to reach the goal while retaining compositionality.

proves over model-based planning and offline RL algorithms across an array of simulated benchmarks as well as a real world robot experiment. RSM outperforms prior methods on the OCBench and RISE Robomimic benchmarks in long horizon planning and planning with visual inputs, as well as showcasing improvements in planning latency and horizon scaling compared to world models. Finally, we show applications of RSM to hierarchical planning and zero-shot reinforcement learning.

2 Preliminaries

We consider a Markov Decision Process (MDP; Puterman, 1990) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $r : \mathcal{S} \to \mathbb{R}$ is a state-based reward function and $\gamma \in [0, 1)$ is the discount factor. At every time step, the agent selects an action $a \in \mathcal{A}$ at state $s \in \mathcal{S}$ according to a policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. The next state evolves according to the transition model $P : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$. We study the offline setting where a fixed dataset $\mathcal{D}$ is collected by some behavior policy μ. The objective is to learn a policy that improves beyond $\mu(a|s)$ over the objective $J(\pi) = \mathbb{E}^\pi[\sum_{t\geq 0} \gamma^k r(s_t)]$.

The *Successor Measure* (SM; Dayan, 1993 of a policy π is defined as the expected discounted state visitation distribution:

$$d^\pi_\gamma(\cdot \mid s, a) = (1-\gamma) \sum_{k\geq 0} \gamma^k \, \mathbb{E}^\pi \left[\delta_{s_{t+k}}(\cdot) \mid s_t = s,\, a_t = a\right]. \tag{1}$$

It satisfies a Bellman recursion:

$$d^\pi_\gamma(\cdot \mid s, a) = (1-\gamma)\delta_s(\cdot) + \gamma \, \mathbb{E}_{\substack{s'\sim P(\cdot|s,a) \\ a'\sim\pi(\cdot|s')}} \left[d^\pi_\gamma(\cdot \mid s', a')\right], \tag{2}$$

which makes it amenable to learning with Temporal Difference (TD; Sutton, 1988).

The *Q-function*, also known as the state-action value, is defined as the expected discounted sum of rewards from taking action a at state s and then following π: $Q^\pi(s, a) = \mathbb{E}^\pi\left[\sum_{k\geq 0} \gamma^k r(s_{t+k}) \mid s_t = s, a_t = a\right]$ and is commonly used to extract policies or rank actions from a proposal distribution. The Q-function is related to the SM by the following equation:

$$Q^\pi(s, a) = \tfrac{1}{1-\gamma} \, \mathbb{E}_{s_f\sim d^\pi_\gamma(\cdot|s,a)}[r(s_f)]. \tag{3}$$

In this work, we use this relation to estimate values for test-time planning.

3 Related Work

Offline Value-based Reinforcement Learning algorithms learn a policy that improves upon the dataset behavior without environment interaction (Levine et al., 2020). These typically estimate a

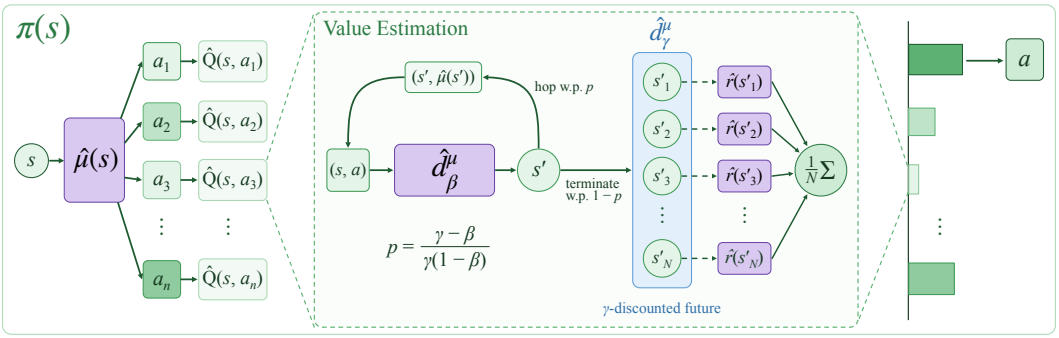


Figure 2: **Recursive Successor Models (RSM).** Planning with RSM involves sampling actions from a proposal policy $\hat{\mu}$ and selecting the action with the highest value. To estimate the value of (s, a) under μ, we predict future states s' at a β-discount and recursively compose them to get the γ-discounted future for some $\gamma > \beta$. The value is the expected reward under these futures states.

value function and extract a policy constrained to the support of the dataset (Kumar et al., 2020; Fujimoto & Gu, 2021; Kostrikov et al., 2022). Values learned with bootstrapped TD targets propagate approximation error during training, which recent work highlights as the main obstacle in scaling TD-based methods to solve long horizon tasks (Park et al., 2026b). Our work takes a different approach to value estimation through generative modeling of the successor measure while avoiding TD entirely.

World Models predict environment dynamics useful for policy learning (Janner et al., 2019; Hafner et al., 2020) and planning (Hafner et al., 2019; Hansen et al., 2022; Zhou et al., 2025). Reasoning over extended horizons with single-step world models requires chaining many predictions, leading to compounding error. This has motivated simultaneous generation of trajectories (Janner et al., 2022; Ajay et al., 2023) and directly predicting consequences of action chunks with multi-step world models (Asadi et al., 2019; Park et al., 2026a). Rather than making predictions at any specific timescale, we mitigate compounding error by learning a policy-dependent world model that enables sampling states directly from the policy's discounted future state distribution, i.e., a *successor model*.

Successor Measures (Dayan, 1993; Blier et al., 2021) describe the discounted distribution of future states visited under a policy. They underlie zero-shot RL via successor features and forward-backward representations (Barreto et al., 2017; 2018; Touati & Ollivier, 2021; Bagatella et al., 2026a), as well as off-policy evaluation via the DICE line of work (Nachum et al., 2019). Closely related to our work, generative approaches learn to sample directly from this distribution (Janner et al., 2020; Thakoor et al., 2022; Farebrother et al., 2025; 2026; Chung et al., 2026). These models are typically trained with TD learning, whereas we train with a fixed supervised objective and compose predictions at test time to enable composition of behaviors.

4 Recursive Successor Models

Our goal is to learn a model from $\mathcal{D}$ that supports planning to improve upon the behavior policy μ. Greedy improvement upon μ has been shown to be simple and effective for a number of tasks (Brandfonbrener et al., 2021; Park et al., 2026b). To accomplish this improvement, we can construct a policy improvement step from a Q function:

$$\pi(s) = \underset{a \in \{a_1, \ldots, a_N\}}{\arg\max} \hat{Q}^{\mu}(s, a), \qquad a_i \sim \hat{\mu}(\cdot \mid s). \tag{4}$$

by selecting the best among N candidates sampled from $\hat{\mu}$, an approximation of the behavior policy μ. To estimate Q^{μ}, we will model the γ-discounted future state occupancy induced by μ, d^{μ}_{γ} in Equation 1, and compute the value following Equation 3 with an approximated reward function, giving rise to an M-sample estimator below:

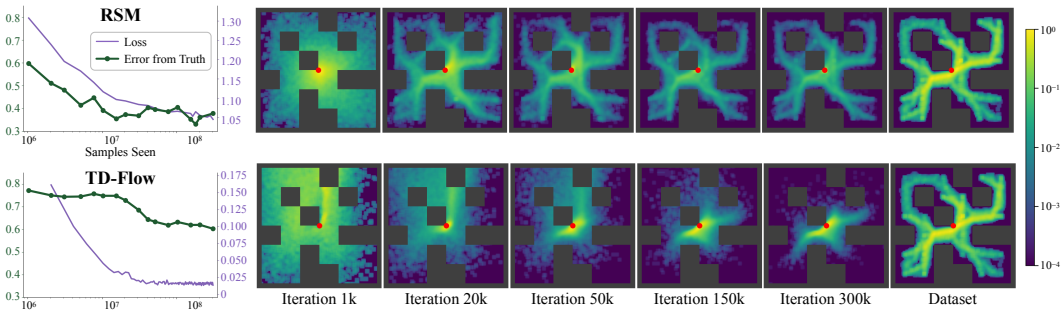


Figure 3: **Sample efficiency of RSM.** We compare the learned visitation distributions (starting from state denoted by the red dot) to the ground truth computed from dataset over training. The left plots show the total variation distance from the ground truth distribution along with the training loss. RSM converges faster than TD-Flow (Farebrother et al., 2025), a method that learns by bootstrapping.

$$\hat{Q}^{\mu}(s,a) = \frac{1}{M(1-\gamma)} \sum_{i=1}^{M} \hat{r}\big(s_f^{(i)}\big), \qquad s_f^{(i)} \overset{\text{i.i.d.}}{\sim} \hat{d}_{\gamma}^{\mu}(\cdot \mid s,a). \tag{5}$$

We choose to estimate the Q-function with an occupancy model in order to receive dense, reward-independent supervision from observed future states, even when task rewards are sparse or absent (Dosovitskiy & Koltun, 2017; Blier et al., 2021). Below, we detail the difficulties in learning d_{γ}^{μ} directly and instead contribute a stable and efficient method for approximating it to support value estimation and policy improvement (Sutton & Barto, 1998).

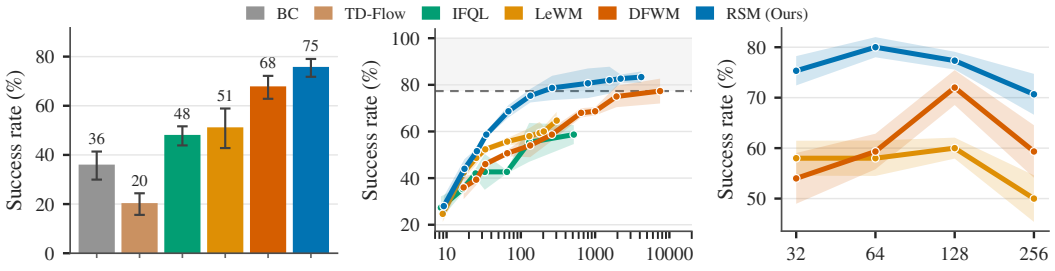
Problem. Estimating d_{γ}^{μ} by approximating P and running one-step model rollouts under $\hat{\mu}$ requires many sequential predictions, increasing latency and compounding errors over long horizons (Janner et al., 2019). Directly modeling d_{γ}^{μ} avoids these rollouts when estimating values via Equation 5. Yet many existing methods use TD updates derived from the Bellman recursion in Equation 2, constructing targets from their own predictions. These updates can propagate approximation errors during training, undermining long-horizon accuracy and stability (van Hasselt et al., 2018). We seek to model d_{γ}^{μ} without one-step rollouts at test time or TD-bootstrapped targets during training.

Approach. We introduce RSM, trained directly on observed futures within each trajectory $\tau \in \mathcal{D}$ using a Monte Carlo generative objective. Conditioned on a discount factor β, RSM produces short-horizon predictions that we recursively compose at test time to sample a longer-horizon γ-discounted future-state distribution (Figure 2). Importantly, this allows us to stitch across various trajectory sub-components, while 1) retaining an exceedingly simple training objective and 2) avoiding autoregressive next state prediction. We describe training, recursive composition, and the value estimation algorithm in Section 4.1, Section 4.2, and Algorithm 1, respectively, and defer training details for $\hat{\mu}$ and $\hat{r}$ to Appendix F.5.

4.1 Monte Carlo Training of RSM

To avoid train-time bootstrapping shown in Equation 2, we train RSM via a typical Monte-Carlo generative modeling objective on states observed in the β-discounted future of each trajectory $\tau \in \mathcal{D}$ of length T for all $\beta \in [0,1]$. Specifically, for a given state-action pair $(s_t, a_t) \in \tau$, we sample $k \sim \text{Geom}(1-\beta)$ and use the truncated future $s_f \in \tau$ as the target where $f = \min(T, t+k)$. This provides a direct Monte Carlo sample from the successor measure $d_{\beta}^{\mu}(\cdot \mid s_t, a_t)$, without needing targets from the model's own predictions. The benefits of this objective are highlighted in Figure 3.

We learn a model of the discount-conditioned SM, $\hat{d}^{\mu}(\cdot \mid s, a, \beta)$, with conditional flow matching (Lipman et al., 2023; Liu et al., 2022; Tong et al., 2024). This successor model is conditioned on actions a, so that a downstream $Q^{\mu}(s,a)$ can be estimated and the best action chosen for test time inference. We train over a log-weighted range of discount factors β so that different successor horizons can be queried at test time, which we use for the recursion described in Section 4.2.



(a) Success rate (b) Inference time (ms) (c) Horizon

Figure 4: **Long-horizon planning.** **(a)** RSM achieves the highest mean success across nine tasks. **(b)** RSM attains higher success at lower inference latency. **(c)** RSM outperforms one- and multi-step world models across horizons (controlled by γ) and degrades more gracefully. See Section 5.1.

For a target future state s_f, we sample $z_0 \sim \mathcal{N}(0, I)$ and $\lambda \sim \mathcal{U}[0, 1]$. We optimize the conditional flow-matching objective at training time:

$$\mathcal{L}(\theta) = \mathbb{E}\Big[\|v_\theta(z_\lambda, \lambda \mid s_t, a_t, \beta) - (s_f - z_0)\|^2\Big], \qquad z_\lambda = (1-\lambda)z_0 + \lambda s_f. \tag{6}$$

At test time, we generate successor samples by initializing $z_0 \sim \mathcal{N}(0, I)$ and integrating the learned velocity field with standard Euler integration.

4.2 Test-time β-recursion

A drawback of eliminating TD bootstrapping is that supervision only maps each state-action pair to future states observed within the same trajectory. Consequently, a successor model trained to solely generate $s_f \in \tau$ from $(s_t, a_t) \in \tau$ cannot by itself compose useful behavior segments from a different trajectory $\tau' \neq \tau$. Such composition of different trajectory segments is desirable for planning, since it allows the agent to construct novel sequences from diverse experiences as depicted in Figure 1as well as Appendix A. Recursively sampling can allow us to combine observed behaviors into sequences not present in any single trajectory. This is because a prediction reaching an intermediate state, s_m, seen in both τ and τ', can be followed by a continuation from τ' or τ if another prediction is made from s_m.

The question which then remains is: how much recursion do we employ for a given β to create a γ-discounted distribution? More precisely, in order for the distribution induced by our recursive chain to estimate the γ-discounted SM, the distribution of timesteps covered by all recursive hops must match $T_\gamma \sim \text{Geom}(1-\gamma)$. If we continue recursion with probability p after each hop the number of chained hops follows $N \sim \text{Geom}_1(1-p)$. Each hop covers an independent temporal offset $T_\beta^{(i)} \sim \text{Geom}_0(1-\beta)$, and the total offset is $T = \sum_{i=1}^{N} T_\beta^{(i)}$. Choosing $p = (\gamma - \beta)/(\gamma(1-\beta))$ gives $T \sim \text{Geom}_0(1-\gamma)$, as proven in Appendix D. Intuitively, this shows that composing SMs associated with a small β should be called more to construct a larger γ-discounted distribution while an already large β would require less. The qualitative effect of β is shown in Appendix 12.

5 Experiments

We conduct experiments to study RSM's planning and behavior composition capabilities in long-horizon manipulation (Sec. 5.1) and learning from suboptimal and play data from pixels (Sec. 5.2). We additionally demonstrate the versatility of RSM by applying it to subgoal generation in goal-conditioned tasks and to zero-shot RL (Sec. 5.3), and include analysis on the inference latency benefits and success rate scaling laws as a function of horizon (Sec. 5.1).

Baselines and planning protocol. We compare behavior cloning (**BC**) without planning against best-of-N action selection with different value estimators. Within each benchmark, all planning methods use some shared behavior-cloned proposal policy, controlling for action proposals while varying value estimation. **IDQL** (Hansen-Estruch et al., 2023) and **IFQL** (Park et al., 2025b) learn scalar critics with TD learning. **TD-Flow** (Farebrother et al., 2025) learns a successor model with TD bootstrapping. We pair it with RSM's planning procedure to isolate the effect of the training objective. **LeWM** (Maes et al., 2026) and **DINO-WM** (Zhou et al., 2025) estimate values from

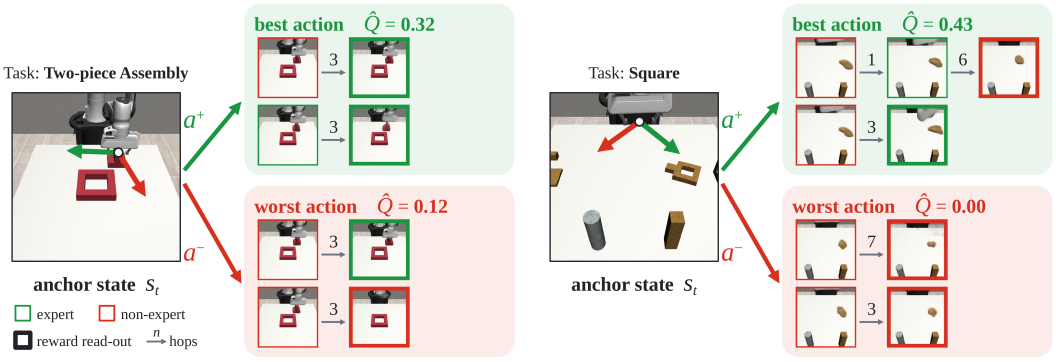


Figure 5: **Qualitative successor samples on RISE.** We visualize latent β-recursion chains generated by RSM starting from a shared anchor state s_t. We show the chains associated with the highest and lowest estimated Q-value actions. Chain images are output by a pixel decoder trained over DINOv2 latents. Color-coding denotes learned reward's expert or non-expert state classification.

one-step world-model rollouts, while **Diffusion Forcing (DF)** (Chen et al., 2024) uses multi-step predictions. Model-based methods learn a reward model. See Appendix 6.

5.1 Long Horizon Manipulation

We evaluate long-horizon planning on OCBench (Anonymous, 2026), an adaptation of OGBench (Park et al., 2025a) designed to better reflect realistic robotic settings, containing noisy and suboptimal manipulation trajectories with higher control frequencies, averaging $4{,}500$ timesteps. Figure 4a reports the aggregate success rate across eight tasks and RSM provides the largest gains over the baselines. Compared with IFQL, RSM avoids learning long-horizon value directly with TD value backup and instead decomposes value estimation into successor prediction and reward estimation, suggesting that this factorization provides a more effective learning signal from offline trajectories. Compared with LeWM and DFWM, RSM does not require autoregressive prediction at each step when predicting future states to evaluate an action, avoiding the compounding prediction errors and inference cost of world-model rollouts. This difference is also reflected in Figure 4b, where RSM achieves higher performance at lower inference latency across the evaluated test-time compute budgets. The configuration at each inference time budget is shown in Appendix F. Finally, RSM outperforms TD-Flow, indicating the superiority of its supervised learning objective. Figure 4 shows that RSM consistently outperforms world models at comparable effective planning horizons (determined by γ in RSM), highlighting its advantage for long-horizon prediction. RSM also maintains its performance better as the planning horizon increases. Full per-task results are provided in Appendix Table 16.

5.2 Planning over visual inputs with human-collected data

Visual manipulation benchmark. We evaluate RSM on seven simulated multistage manipulation tasks (Mandlekar et al., 2021) from RGB images and proprioception, using human-collected data following RISE (Huang et al., 2025). The **suboptimal (sub.)** setting augments expert demonstrations with failed or partial executions. The **coverage (cov.)** setting adds undirected play and evaluates broader initial object configurations than those in the expert data. These settings test whether non-expert data can improve task completion from states poorly covered by expert demonstrations. Training rewards are 1 for expert transitions and 0 otherwise, while evaluation uses MimicGen-based task success. Data collection and implementation details are provided in Appendix E.

Baselines and training data. We include behavior-cloned diffusion policies trained on expert data (**BC**) or the union of expert and non-expert data (**BC-u**), both without planning. To isolate the effect of value estimation, all planning methods share the same RISE proposal policy: a diffusion policy with spectral normalization and action data augmentation. The proposal policy and all value estimators are trained on the union dataset. All critic methods condition on DINOv2 latents, and all

Table 1: **RISE offline RL comparison over suboptimal and play data.** RSM performs better or on-par with behavior cloning, model-based planning, and offline RL baselines. We report success rates over 3 seeds (random actor noise, environment conditions) with 50 rollouts each.

Task	BC	BC-u	IDQL	IFQL	TD-Flow	LeWM	DINO-WM	DF	RSM
Square (Sub.)	14.4±2.2	48.0±1.4	43.6±1.2	52.0±3.8	50.0±4.8	48.7±1.4	50.0±1.4	50.7±2.7	**54.0±1.3**
Square (Cov.)	11.0±1.0	3.1±0.6	3.7±0.2	2.0±0.8	7.1±1.1	3.1±0.6	13.3±3.3	6.0±0.4	**20.9±2.6**
Piece (Sub.)	13.8±2.0	33.0±1.7	22.9±6.1	23.6±1.0	30.2±1.9	29.8±1.9	41.1±1.7	28.0±1.0	**47.1±3.5**
Piece (Cov.)	18.9±1.2	25.3±2.2	6.0±1.7	21.1±1.9	16.0±1.4	24.0±1.4	37.6±3.7	20.7±0.7	**48.7±0.7**
Mug (Sub.)	14.9±1.7	4.9±0.2	16.2±1.0	18.4±1.0	19.8±0.9	17.3±1.4	18.4±1.0	19.6±0.6	**23.1±2.1**
Thread (Cov.)	26.3±1.2	1.3±0.2	12.2±1.4	8.4±1.2	7.1±1.4	10.2±0.4	**31.6±1.6**	18.2±1.5	25.8±1.1
Piece-tip (Cov.)	2.0±1.0	1.0±0.2	2.9±0.6	3.3±0.4	1.1±0.6	3.8±0.8	5.1±0.6	1.3±0.0	**7.3±0.4**
Mean	14.5±2.8	16.7±7.1	15.4±5.4	18.4±6.5	18.8±6.4	19.6±6.1	28.2±6.2	20.6±6.1	**32.4±6.6**

Table 2: **OGBench high-level planning.** Success rate per category (number of tasks in parentheses). For each seed, success is averaged over the category's tasks; we report the mean ± sample standard deviation over 3 seeds.

Observation	Category	IQL	IFQL	SWM	SWM+V	HIQL	SHARSA	TD-Flow	RSM
`state`	navigation (8)	54 ± 6	67 ± 3	47 ± 1	59 ± 1	76 ± 2	**89** ± 1	75 ± 4	**89** ± 2
	manipulation (4)	58 ± 4	55 ± 1	54 ± 1	58 ± 3	31 ± 3	28 ± 1	64 ± 2	**68** ± 1
`visual`	navigation (3)	77 ± 2	42 ± 1	—	—	80 ± 5	**85** ± 5	75 ± 2	**86** ± 5
	manipulation (3)	20 ± 4	39 ± 1	—	—	11 ± 5	**44** ± 8	38 ± 5	**46** ± 3

world-model methods condition and generate DINOv2 latents. Implementation details and hyperparameter sweeps are provided in Appendix E.

Results. RSM achieves the highest reported success rate on six of seven task–dataset settings (Table 1). Averaged across settings, RSM achieves 32.4% success, compared with 28.2% for the strongest overall baseline (DINO-WM) and 16.7% for the strongest BC baseline (BC-u). RSM substantially improves mean success over the per-task best of BC and BC-u in both suboptimal and coverage data regimes, with relative gains of 29.5% and 59.0%, respectively. This indicates RSM can stitch from distant play and failure states back onto the expert state distribution (Fig. 5).

The baseline comparisons suggest that how the future is modeled matters beyond simply using a successor representation or predicting multiple steps ahead. TD-Flow performs similarly to scalar-critic IFQL on average, whereas RSM exceeds TD-Flow on every setting. This is consistent learning successors directly with supervised learning being superior to TD learning. Our Diffusion Forcing baseline also predicts longer horizon (multi-step) futures, but has a higher-dimensional and potentially more difficult modeling problem, which can explain DF's weaker planning performance compared to RSM. We also note that LeWM is significantly worse than DINO-WM, which alludes to the difficulty of simultaneously learning the representation and world model.

5.3 Additional Applications

Subgoal Generation. We show that test-time planning with RSM yields useful subgoals for goal-conditioned control, evaluated on OGBench (Park et al., 2025a). Subgoal generation demands less of RSM than value estimation: rather than estimating an expectation over many sampled trajectories, it suffices to find a single state sequence that reaches the goal. We propose a simple search procedure that recursively samples state sequences from a state-goal RSM $d^{\mu}_{\beta}(\cdot \mid s, g)$, stopping once a learned reward model indicates the goal is reached. A state from this sequence is passed as a subgoal to a goal-conditioned low-level policy (Alg. 2, App.). Since hop count correlates with path length, halting at the first goal-reaching sequence favors shorter paths. IQL (Kostrikov et al., 2022) and IFQL (Hansen-Estruch et al., 2023) serve as base policies for the hierarchical methods: HIQL (Park et al., 2023), SHARSA (Park et al., 2026b) and a subgoal generator trained with the TD-Flow (Farebrother et al., 2025) objective. We additionally compare to a single-step world model (SWM). Aggregated results are presented in Table 2. Despite its simplicity, RSM subgoal generation matches or outperforms all baselines, significantly surpassing TD-Flow which only differs from it in the SM learning objective. See Apppendix H for full experiment details.

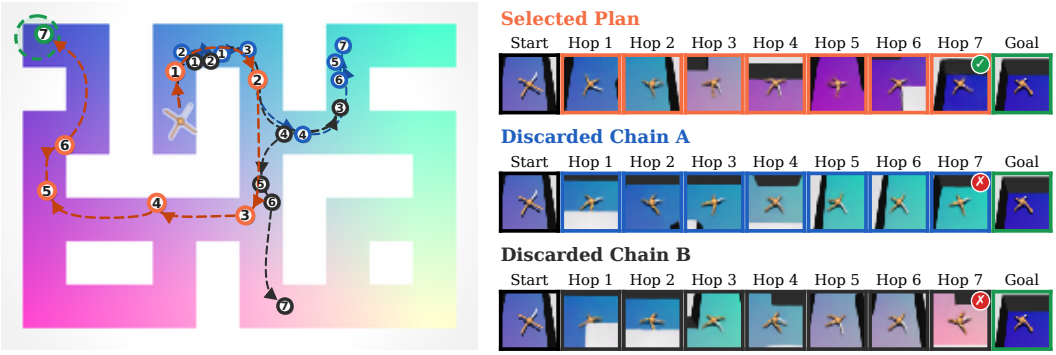


Figure 6: **RSM for subgoal generation on** `visual-antmaze-large-navigate`. *Left*: RSM plans are presented on top of the maze. The successful goal-reaching plan is depicted in orange, and two example failed plans are depicted in blue and black. *Right*: Decoded RSM-generated states corresponding to the plans depicted on the maze.

Table 3: **Zero-shot reinforcement learning results**. Aggregated mean return over all task suites in each domain. Full results in Table 18.

Domain	Laplacian	BYOL-γ	TD-JEPA	ICVF	HILP	FB	One-Step FB	RSM-ZS (ours)
ExORL (4 suites)	204.5	213.3	243.8	343.8	220.3	231.8	356.0	**456.8**
OGBench (4 suites)	5.5	16.3	8.0	18.3	25.5	12.3	13.0	**32.6**

Zero-Shot Reinforcement Learning (Borsa et al., 2019; Touati & Ollivier, 2021; Touati et al., 2023) is an emerging RL paradigm where the learner is given an offline transition dataset at training time and needs to adapt to arbitrary downstream reward at test time. In Appendix I, we extend RSM to the zero-shot RL setting by generating the successor features in addition to future states. We evaluate the zero-shot RL method, RSM-ZS on two benchmarks: ExoRL (Denis Yarats, 2022) and OGBench (Park et al., 2025a). Each benchmark contains a number of tasks suites, and each suite provides an offline transition dataset and a number of downstream tasks to transfer to. We use the same evaluation protocol as (Zheng et al., 2026) and compare to baselines from zero-shot RL literature (Wu et al., 2019; Lawson et al., 2026; Ghosh et al., 2023; Park et al., 2024; Bagatella et al., 2026b; Touati & Ollivier, 2021; Zheng et al., 2026). Details on environments and baselines are provided in I.2. Table 3 shows the aggregated results. Across both benchmarks, multi-hop RSM-ZS outperforms baseline methods by a margin. This shows that modeling short-horizon cumulative features and composing them into successor feature via multi-hop composition at test time effectively enables long-horizon planning with simple best-of-n selection.

5.4 Real Robot Experiments

Can RSM enable long-horizon planning from suboptimal play data on real-world robotic hardware? We evaluate RSM on a Franka Research 3 (FR3) in a kitchen environment. We collect 60.51 minutes of structured play data comprising pick-and-place, object reorientation, and pouring interactions, and use this data to train the successor and reward models. At test time, $\pi_{0.5}$ serves as the proposal policy for best-of-N planning, with RSM used to score and select among candidate actions. We evaluate the resulting system on a long-horizon kitchen task requiring the robot to place a piece of cheese in a pan, grasp the pan and pour the contents of the pan into a bowl.

Results. Results illustrated in Fig. 7. Across two sampled initial-state configurations from the play data, the $\pi_{0.5}$ behavior-cloning baseline achieves 0% success. In contrast, RSM achieves 40% success starting from Stage 2 and 10% end-to-end task success (starting from Stage 1) which is a 60 second horizon task consisting of 3000 control steps.



Figure 7: Real-world task completion starting from different initial state distributions (denoted "stages" above). Left: Success rates of BC and RSM when the scene is initialized in the configuration depicted in Stage 1 or Stage 2. Both methods share the same final goal, shown in Stage 4. Right: Selected frames from a successful RSM rollout illustrate progression through the task.

5.5 Analysis

Does greater action diversity help? We first test whether RSM can benefit from additional test-time compute by varying the number of sampled action candidates. Although best-of-N planning should improve with a larger candidate set, biased value estimates may prevent the planner from ranking candidates reliably. Figure 10a shows that both RSM and DFWM benefit from additional candidates, indicating that their value estimates remain informative as action diversity increases. In contrast, LeWM and IFQL gain little from additional candidates, suggesting that world-model compounding error and TD-based value estimation limit their ability to exploit greater action diversity at test time.

How does recursion depth affect RSM? We vary the successor discount β on `chamber-hard-task1` in OCBench Anonymous (2026) (Section 5.1), which controls the effective recursion depth. Figure 10b shows an intermediate optimum. Too little recursion limits long-horizon reasoning, while too much recursion compounds modeling error. However, RSM still outperforms IFQL and LeWM at both extremes, indicating that its performance is relatively robust to the choice of β.

6 Discussion

We introduce Recursive Successor Models (RSM), a scalable recipe for learning long-horizon occupancy models from offline datasets. We show that Monte-Carlo supervision and test-time recursion provide a stable way to estimate behavior value for downstream planning. The empirical success of RSM implies that a simple greedy approach to planning with the behavior value function is a strong alternative to multi-step policy improvement. β represents a dial to control the extent to which we decompose long-horizon future into short-hops, thus controlling an important tradeoff in variance, modeling bias, stitching capabilities, and compounding errors. To achieve strong results, β is often set fairly high, implying that many tasks naturally require composition over longer segments of time.

Limitations. RSM planning requires a significant amount of test-time compute since action selection involves rolling out multiple samples per-action. That said, it has significantly less sequential dependencies than planning with single-step world models, making it more parallelizable. Although RSM rollouts require far fewer autoregressive steps than single-step world models, prediction errors still compound over long horizons. Finally, our planning framework estimates the behavior value rather than the optimal value, which limits it to one-step policy improvement. This has proved effective in all of our experiment settings but may not be sufficient when learning from highly suboptimal data.

Future work. RSM suggests several directions for future work. In robotics, its strong performance and low inference latency make it promising for real-time planning. The successor-model objective could also serve as an auxiliary training signal for VLA policies, analogous to the role of world-action modeling in recent VLA methods. Finally, because successor modeling does not require temporally smooth state transitions, it may be particularly well suited to contact-rich dynamics where standard world models can struggle.

AI USE STATEMENT

In this work, we used generative AI tools for helping implement code for different methods, assisting in writing concisely, assisting in figure generation, providing ingredients to help write proofs, helping to develop conceptual frameworks from prior works, refining hypotheses about method performance, helping interpret experimental results, and conducting literature review. All interpretations and writing made by AI were verified and all work and output was reviewed. We have not used generative AI tools for generating synthetic data, reformatting or cleaning data, and autonomously conducting data analysis. Assisting with translation is not applicable to this work.

Additionally, we used generative AI tools for modifying figures, editing software, creating artifacts, summarize existing literature, brainstorming, searching for papers, managing and launching experiment jobs, and editing paragraphs or sentences for better clarity. We have reviewed all AI-assisted work. In particular, all LLM-generated code was verified and tested, LLM ideas were cross referenced with existing work, interpretations of results were primarily done by humans with AI summaries verified by raw numerical reporting, literature reviews were to source papers and the source content was manually read, and all figures/writing merely used AI as an assistive tool. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This research adheres to the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of our method, training procedure, experimental setup, and evaluation protocol in the main paper and appendix. Additional implementation details, hyperparameters, and ablations are included in the appendix.

REFERENCES

Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B. Tenenbaum, Tommi S. Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision making? In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=sP1fo2K9DFG.

Anonymous. OCBench: a controllable benchmark for studying behavioral cloning, 2026. URL https://openreview.net/forum?id=bTZVvwj6R6. Submitted to ICLR 2027.

Kavosh Asadi, Dipendra Misra, Seungchan Kim, and Michel L Littman. Combating the compounding-error problem with a multi-step model. *arXiv preprint arXiv:1905.13320*, 2019.

Marco Bagatella, Matteo Pirotta, Ahmed Touati, Alessandro Lazaric, and Andrea Tirinzoni. Td-jepa: Latent-predictive representations for zero-shot reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 36211–36244, 2026a.

Marco Bagatella, Matteo Pirotta, Ahmed Touati, Alessandro Lazaric, and Andrea Tirinzoni. TD-JEPA: Latent-predictive representations for zero-shot reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=SzXDuBN8M1.

André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado Van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 501–510. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/barreto18a.html.

Léonard Blier, Corentin Tallec, and Yann Ollivier. Learning successor states and goal-dependent values: A mathematical viewpoint. *arXiv preprint arXiv:2101.07123*, 2021.

Diana Borsa, Andre Barreto, John Quan, Daniel J. Mankowitz, Hado van Hasselt, Remi Munos, David Silver, and Tom Schaul. Universal successor features approximators. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1VWjiRcKX.

David Brandfonbrener, William F. Whitney, Rajesh Ranganath, and Joan Bruna. Offline rl without off-policy evaluation, 2021. URL https://arxiv.org/abs/2106.08909.

Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2024.

Hojun Chung, Junseo Lee, and Songhwai Oh. Offline reinforcement learning with universal horizon models. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=GFmJ5XcGWl.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993. doi: 10.1162/neco.1993.5.4.613.

Hao Liu Michael Laskin Pieter Abbeel Alessandro Lazaric Lerrel Pinto Denis Yarats, David Brandfonbrener. Don't change the algorithm, change the data: Exploratory data for offline reinforcement learning. *arXiv preprint arXiv:2201.13425*, 2022.

Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future, 2017. URL https://arxiv.org/abs/1611.01779.

Jesse Farebrother, Matteo Pirotta, Andrea Tirinzoni, Remi Munos, Alessandro Lazaric, and Ahmed Touati. Temporal difference flows. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=j6H7c3aQyb.

Jesse Farebrother, Matteo Pirotta, Andrea Tirinzoni, Marc G Bellemare, Alessandro Lazaric, and Ahmed Touati. Compositional planning with jumpy world models. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=cwztmjMQ77.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

Dibya Ghosh, Chethan Anand Bhateja, and Sergey Levine. Reinforcement learning from passive data via latent intentions. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11321–11339. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/ghosh23a.html.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/2de5d16682c3c35007e4e92982f1a2ba-Paper.pdf.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pp. 2555–2565. PMLR, 2019.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1lOTC4tDS.

Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. In *ICML*, 2022.

Philippe Hansen-Estruch, Ilya Kostrikov, Michael Janner, Jakub Grudzien Kuba, and Sergey Levine. Idql: Implicit q-learning as an actor-critic method with diffusion policies. *arXiv preprint arXiv:2304.10573*, 2023.

Kevin Huang, Rosario Scalise, Cleah Winston, Ayush Agrawal, Yunchu Zhang, Rohan Baijal, Markus Grotz, Byron Boots, Benjamin Burchfiel, Masha Itkina, Paarth Shah, and Abhishek Gupta. Using non-expert data to robustify imitation learning via offline reinforcement learning, 2025. URL https://arxiv.org/abs/2510.19495.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

Michael Janner, Igor Mordatch, and Sergey Levine. Gamma-models: Generative temporal difference learning for infinite-horizon prediction. *Advances in neural information processing systems*, 33: 1724–1735, 2020.

Michael Janner, Yilun Du, Joshua B. Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, 2022.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=68n2s9ZJWF8.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in neural information processing systems*, 33:1179–1191, 2020.

Daniel Lawson, Adriana Hugessen, Charlotte Cloutier, Glen Berseth, and Khimya Khetarpal. Self-predictive representations for combinatorial generalization in behavioral cloning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=FkeURAdA0h.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Chenhao Li, Andreas Krause, and Marco Hutter. Robotic world model: A neural network simulator for robust policy optimization in robotics. *arXiv preprint arXiv:2501.10100*, 2025.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling, 2023. URL https://arxiv.org/abs/2210.02747.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

Lucas Maes, Quentin Le Lidec, Damien Scieur, Yann LeCun, and Randall Balestriero. Leworld-model: Stable end-to-end joint-embedding predictive architecture from pixels. *arXiv preprint arXiv:2603.19312*, 2026.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.

Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. *Advances in neural information processing systems*, 32, 2019.

Kwanyoung Park, Seohong Park, Youngwoon Lee, and Sergey Levine. Scalable offline model-based rl with action chunks. In *International Conference on Learning Representations*, volume 2026, pp. 9599–9623, 2026a.

Seohong Park, Dibya Ghosh, Benjamin Eysenbach, and Sergey Levine. Hiql: Offline goal-conditioned rl with latent states as actions. *Advances in Neural Information Processing Systems*, 36:34866–34891, 2023.

Seohong Park, Tobias Kreiman, and Sergey Levine. Foundation policies with Hilbert representations. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 39737–39761. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/park24g.html.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. OGBench: Benchmarking offline goal-conditioned RL. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=M992mjgKzI.

Seohong Park, Qiyang Li, and Sergey Levine. Flow q-learning. *arXiv preprint arXiv:2502.02538*, 2025b.

Seohong Park, Kevin Frans, Deepinder Mann, Benjamin Eysenbach, Aviral Kumar, and Sergey Levine. Horizon reduction makes rl scalable. *Advances in Neural Information Processing Systems*, 38:8350–8389, 2026b.

Martin L. Puterman. Chapter 8 markov decision processes. In *Stochastic Models*, volume 2 of *Handbooks in Operations Research and Management Science*, pp. 331–434. Elsevier, 1990. doi: https://doi.org/10.1016/S0927-0507(05)80172-0. URL https://www.sciencedirect.com/science/article/pii/S0927050705801720.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3 (1):9–44, 1988.

R.S. Sutton and A.G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998. doi: 10.1109/TNN.1998.712192.

Shantanu Thakoor, Mark Rowland, Diana Borsa, Will Dabney, Rémi Munos, and André Barreto. Generalised policy improvement with geometric policy composition. In *International Conference on Machine Learning*, pp. 21272–21307. PMLR, 2022.

Alexander Tong, Kilian Fatras, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport, 2024. URL https://arxiv.org/abs/2302.00482.

Ahmed Touati and Yann Ollivier. Learning one representation to optimize all rewards. *Advances in Neural Information Processing Systems*, 34:13–23, 2021.

Ahmed Touati, Jérémy Rapin, and Yann Ollivier. Does zero-shot reinforcement learning exist? In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=MYEap_OcQI.

Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad, 2018. URL https://arxiv.org/abs/1812.02648.

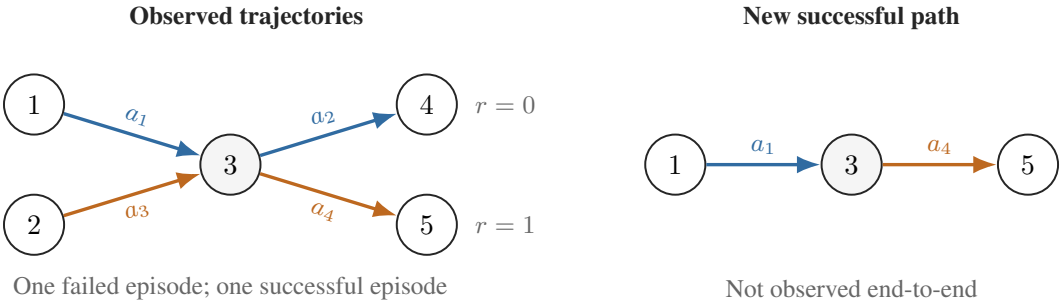
Yifan Wu, George Tucker, and Ofir Nachum. The laplacian in RL: Learning representations with efficient approximations. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJlNpoA5YQ.

Chongyi Zheng, Royina Karegoudra Jayanth, and Benjamin Eysenbach. Can we really learn one representation to optimize all rewards? *arXiv preprint arXiv:2602.11399*, 2026.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. Dino-wm: World models on pre-trained visual features enable zero-shot planning. In *Forty-second International Conference on Machine Learning*, 2025.

Chuning Zhu, Xinqi Wang, Tyler Han, Simon Shaolei Du, and Abhishek Gupta. Distributional successor features enable zero-shot policy optimization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

A STITCHING EXAMPLE



One failed episode; one successful episode

Not observed end-to-end

Why stitching changes the value estimate. Consider two recorded trajectories $1 \xrightarrow{a_1} 3 \xrightarrow{a_2} 4$ and $2 \xrightarrow{a_3} 3 \xrightarrow{a_4} 5$ in a deterministic two-step problem, with reward 1 upon entering terminal state 5 and zero otherwise (state 4 is also terminal). Tabular Monte Carlo return regression gives $\widehat{Q}_{\mathrm{MC}}(1, a_1) = 0 + \gamma \cdot 0 = 0$, since the only recorded continuation from state 1 fails. The empirical behavioral policy instead assigns probability $1/2$ to each observed action at the shared state 3. Composing the transition $1 \to 3$ with these continuations gives $\widehat{Q}^{\hat{\mu}}_{\mathrm{stitch}}(1, a_1) = 0 + \gamma\big(\frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1\big) = \gamma/2$. Thus behavioral stitching assigns positive probability to the unseen full path $1 \to 3 \to 5$, without maximizing over future actions during value estimation. A planner that then chooses a_4 at state 3 executes this path with return γ.

B SUCCESSOR-MODELS PRIOR WORK COMPARISON

Table 4: **Comparison of generative successor models.** We compare γ-MVE, TD-Flow's generalized policy improvement (GPI), Jumpy's compositional planner, and RSM's candidate-action planner.

Method	**Model training**	**Recursion / composition**	**Value estimation / action selection**
γ-models (Janner et al., 2020)	Generative TD, bootstrapped model predictions.	Bellman bootstrap, weighted successor rollouts for value expansion.	Weighted rewards plus terminal value, train a SAC actor–critic.
TD-Flow (Farebrother et al., 2025)	TD flow matching, bootstrapped paths and vector fields.	Bellman bootstrap, direct successor queries in the GPI planner.	Mean sample reward, no terminal critic, select a policy via GPI.
Jumpy World Models (Farebrother et al., 2026)	TD flow matching plus cross-horizon consistency.	Cross-discount training targets, policy-sequence composition at test time.	Weighted rewards, search over policy sequences / subgoals and replan.
RSM (ours)	**Supervised** flow matching on observed futures, no bootstrap.	**Inference-only** β-to-γ composition with geometric stopping.	Mean endpoint reward, no terminal critic, rank initial actions with a fixed continuation policy and replan.

RSM's stopping rule samples the mixture in Janner et al. (2020, Theorem 1). It is not a new composition identity. TD-Flow also discusses Monte Carlo fitting, but its row describes the proposed TD-based method.

C Recursive Value Estimation Algorithm

Algorithm 1 RSM Value Estimation

Require: State s, action a, reward r
Require: Model $\hat{d}_\theta$, policy π
Require: Discounts $\beta < \gamma$, samples N
Require: Depth cap $L \geq 1$

$p \leftarrow \frac{\gamma - \beta}{\gamma(1-\beta)}$ ▷ Continuation probability
function SAMPLE(s, a, ℓ) ▷ ℓ: remaining hop budget
  $z \sim \hat{d}_\theta(\cdot \mid s, a, \beta)$ ▷ Sample a short-horizon future
  $b \sim \text{Bernoulli}(1-p)$
  if $b = 1$ **or** $\ell = 1$ **then** ▷ Stop or reach depth cap
    return z
  end if
  $a' \leftarrow \pi(z)$ ▷ Choose an action at the sampled state
  return SAMPLE($z, a', \ell - 1$) ▷ Recursively compose another hop
end function
for $i = 1$ **to** N **do**
  $z \leftarrow$ SAMPLE(s, a, L) ▷ Start an independent chain
  $R_i \leftarrow r(z)$ ▷ Score the returned state
end for
return $\widehat{Q}(s,a) = \frac{\sum_{i=1}^{N} R_i}{(1-\gamma)N}$ ▷ Average rewards and rescale

D Proof of Recursive Rule

Let $0 < \beta \leq \gamma < 1$. Each successor sample corresponds to an independent temporal offset $T_\beta^{(i)} \sim \text{Geom}_0(1-\beta)$, where

$$\Pr(T_\beta^{(i)} = k) = (1-\beta)\beta^k, \qquad k = 0, 1, \ldots. \tag{7}$$

After each hop, we independently continue with probability p or stop with probability $1-p$. Equivalently, we draw $B_i \sim \text{Bernoulli}(1-p)$ and stop when $B_i = 1$. The number of hops is therefore

$$N = \min\{n \geq 1 : B_n = 1\} \sim \text{Geom}_1(1-p), \qquad \Pr(N = n) = (1-p)p^{n-1}. \tag{8}$$

We assume that the stopping decisions are independent of all temporal offsets and that the chain has no depth cap. The probability generating functions of one temporal offset and the hop count are

$$G_\beta(x) = \mathbb{E}[x^{T_\beta^{(i)}}] = \frac{1-\beta}{1-\beta x}, \qquad G_N(x) = \mathbb{E}[x^N] = \frac{(1-p)x}{1-px}. \tag{9}$$

The total temporal offset is $T = \sum_{i=1}^{N} T_\beta^{(i)}$. By independence,

$$G_T(x) = \mathbb{E}\big[G_\beta(x)^N\big] \tag{10}$$
$$= G_N(G_\beta(x)) \tag{11}$$
$$= \frac{(1-p)(1-\beta)}{1-p(1-\beta)-\beta x}. \tag{12}$$

To match the target distribution $T_\gamma \sim \text{Geom}_0(1-\gamma)$, choose

$$p = \frac{\gamma - \beta}{\gamma(1-\beta)}. \tag{13}$$

For the stated parameter range, $0 \leq p < 1$, and

$$1 - p(1-\beta) = \frac{\beta}{\gamma}, \qquad (1-p)(1-\beta) = \frac{\beta(1-\gamma)}{\gamma}. \tag{14}$$

Substituting gives

$$G_T(x) = \frac{\beta(1-\gamma)/\gamma}{\beta/\gamma - \beta x} = \frac{1-\gamma}{1-\gamma x} = G_{T_\gamma}(x). \tag{15}$$

Since probability generating functions uniquely determine distributions on the nonnegative integers, $T \sim \mathrm{Geom}_0(1-\gamma)$. Thus, continuing with probability p and stopping with probability $1-p$ recovers the desired γ-discounted temporal horizon.

E RISE Experiment Details

RISE visual manipulation benchmark. We independently reproduce the RISE experimental setup, using our own datasets and reward/success function implementations due to lack of data and partial code release from the RISE authors. Thus, some baseline numbers may differ from the original paper.

RISE methods and evaluation. All planning methods differ largely in how the Q-value used for action-selection is estimated:

1. **Scalar methods (IDQL, IFQL).** A critic Q_θ is trained offline by one-step TD with expectile regression (IQL), and candidates are scored directly: $\hat{Q}(s,a) = Q_\theta(s,a)$.
2. **One-step world models (DINO-WM, LeWM).** A latent dynamics model $z_{k+1} = f_\theta(z_k, a_k)$ is rolled out autoregressively for K steps from z_0, the encoding of s, with $a_0 = a$ and each later action sampled from a stochastic continuation policy, $a_k \sim \pi_c(\cdot \mid z_k)$ for $k \geq 1$. The value is the discounted sum of predicted rewards along the rollout, averaged over M independent rollouts,

$$\hat{Q}(s,a) = \frac{1}{M} \sum_{m=1}^{M} \sum_{k=1}^{K} \gamma^k \, \hat{r}\big(z_k^{(m)}\big). \tag{16}$$

3. **Multi-step world model (Diffusion Forcing).** The whole future, including the continuation actions, is generated jointly rather than one step at a time, $(z_{1:K}, a_{1:K-1}, \hat{r}_{1:K}) \sim p_\theta(\cdot \mid z_0, a_0{=}a)$, and the discounted reward sum is averaged over M sampled futures,

$$\hat{Q}(s,a) = \frac{1}{M} \sum_{m=1}^{M} \sum_{k=1}^{K} \gamma^k \, \hat{r}_k^{(m)}. \tag{17}$$

4. **TD-Flow** learns a successor model and is evaluated with the same β-recursion and readout as RSM (Section 4); it differs only in training, where the successor is regressed onto a bootstrapped target, $s^+ \sim (1-\beta)\,\delta_{s'} + \beta\,\bar{m}_\beta(\cdot \mid s', a')$, with $\bar{m}_\beta$ a target network, instead of onto observed future states.

For every model-based method, actions after the candidate come from a stochastic continuation policy: the behaviour-cloned policy π_c, which the one-step world models sample at every step and the successor methods sample at every hop, or, for Diffusion Forcing, the model's own jointly sampled actions. Each of the M rollouts (chains) is therefore an independent sample of the future under that policy. The horizon enters through γ: world models truncate the rollout at $K \approx 1/(1-\gamma)$ steps, while successor methods take a stochastic number of hops per chain. For all methods the horizon enters through γ; world models truncate the rollout at $K \approx 1/(1-\gamma)$ steps. Successor methods sample a stochastic number of recurrences (hops) per chain.

Hyperparameters. Table 5 lists fixed settings, searched values, and the number of configurations evaluated per task. Table 6 gives the selected settings, chosen by mean success over 3 training seeds $\times$ 3 evaluation seeds. The sample count M is the number of independent futures averaged per candidate, and H is the world-model rollout horizon in environment steps. For RSM and TD-Flow, β controls the successor-query horizon, while the value discount is fixed at $\gamma = 0.99$.

Table 5: **RISE hyperparameter search.** Configuration counts are per task.

Method	Fixed settings	Searched values	Configs.
BC, BC-u	RISE recipe	None	1
IDQL	Expectile 0.9; $\gamma = 0.99$	None	1
IFQL	Expectile 0.5	$\gamma \in \{0.95, 0.98, 0.99, 0.995, 0.9999\}$	5
RSM	$\gamma = 0.99$	$\beta \in \{0.80, 0.85, 0.90, 0.95, 0.98\}$; $M \in \{8, 16, 64\}$; readout: prob or logit; Euler steps $\{4, 8, 16\}$; model size, seed, and checkpoint	20–30
TD-Flow	$\gamma = 0.99$; $M = 16$	$\beta \in \{0.90, 0.95\}$; readout: prob or logit; model size, seed, and checkpoint	2–3
DINO-WM	$M = 1$	$H \in \{10, 20, 50, 100, 200\}$	5
LeWM	$M = 16$	$H \in \{10, 20, 50, 100, 200\}$	5
DF	—	$H \in \{10, 20, 50, 100, 200\}$; $M \in \{8, 64\}$	≤ 10

Table 6: **Selected RISE settings** for the results in Table 1.

	RSM						TD-Flow		
Task	β	M	Readout	Size	Hops measured	Hops expected	β	Readout	Size
Square (Sub.)	0.98	16	logit	M	1.02	1.02	0.90	logit	M
Square (Cov.)	0.90	16	prob	L	9.98	10.00	0.95	prob	L
Piece (Sub.)	0.95	8	prob	M	4.21	4.21	0.90	prob	M
Piece (Cov.)	0.98	16	prob	L	1.01	1.02	0.95	prob	L
Mug (Sub.)	0.98	16	logit	M	1.01	1.02	0.95	prob	M
Thread (Cov.)	0.98	8	prob	M	1.03	1.02	0.90	prob	M
Piece-tip (Cov.)	0.95	16	prob	M	4.20	4.21	0.95	prob	M

	IFQL	DINO-WM	LeWM	DF	
Task	γ	H	H	H	M
Square (Sub.)	0.995	200	50	100	8
Square (Cov.)	0.95	200	200	100	8
Piece (Sub.)	0.995	200	200	200	8
Piece (Cov.)	0.9999	10	200	20	8
Mug (Sub.)	0.9999	50	200	10	8
Thread (Cov.)	0.9999	200	100	50	64
Piece-tip (Cov.)	0.9999	200	200	20	8

Readout: prob uses the reward model's expert probability and logit uses its pre-sigmoid score for action ranking. *Hops:* each chain uses $1 +$ hops model queries. Model sizes M and L differ only in width. TD-Flow uses $M = 16$ samples per candidate.

Table 7: Composition of the offline data for the six tasks used in the main paper.

	block-double 2	block-quad. 2	chamber-hard 1	chamber-hard 2	chamber-med. 1	chamber-med. 2
Episodes	30,006	96,868	96,957	100,035	30,006	30,006
Ending in success	87%	61%	41%	42%	58%	43%
Controller mistake prob.	0.1	0.1	0.2	0.2	0.2	0.2
Transitions (M)	45.9	390.1	488.6	489.9	128.9	134.3

F OCBENCH EXPERIMENT DETAILS

Two discount factors appear below and are kept distinct. We write γ for a discount applied once per *environment step*, and γ_c for a discount applied once per *prediction* of a model that advances the world by a whole chunk of actions, i.e. once per C environment steps.

F.1 ENVIRONMENT

Every task is a MuJoCo tabletop manipulation scene driven through the Gymnasium interface, in which a UR5e robot arm with a parallel-jaw gripper acts on a table. All tasks are single-task and reward-based: the environment supplies a scalar reward, and no goal observation or goal image.

Actions. One environment step takes a 7-dimensional action. A decision emits a chunk of C consecutive actions ($C = 24$ in the block family, 32 in the chamber and hanoi families), flattened into a single vector and executed open loop before the policy is called again. Actions are normalised per dimension, mapping the 1st-to-99th percentile range of the training data onto $[-1, 1]$, then clipped at ± 5 and mapped back to the environment's range. One decision therefore advances the environment by C steps.

Observations. Observations concatenate the arm's state with the state of the manipulable objects, so their dimension grows with the number of objects: 22 to 65 after preprocessing (Table 8). Six dimensions, the arm's joint velocities, are removed from both the environment and the training data, identically for every method.

Reward and termination. Reward is sparse: 1 on the transition that completes the task and 0 otherwise. The episode ends when the task is completed, and bootstrapped value targets are masked at that transition; otherwise it runs to a per-task limit of 1,000 to 7,000 environment steps. Success is recorded per episode.

F.2 OFFLINE DATA

Each task comes with one fixed offline dataset, stored as a set of shards, and every method trained on a task uses the same shards. Per-task sizes are given in Table 8. The demonstrations come from a scripted controller that makes a mistake with a fixed probability (Table 7).

Batch construction. One data-loading implementation serves every method. Transition indices are drawn uniformly from the shards resident in memory; three shards are held at a time and replaced every 10^4 gradient steps. Indices are clamped at an episode's final transition, so an action chunk that would run past the end repeats the final action rather than continuing into the next episode. For methods that bootstrap from a state n steps ahead, with T the index of the final transition, r the reward and m the termination mask,

$$k_t = \min(n,\, T - t + 1), \qquad R_t = \sum_{j=0}^{k_t - 1} \gamma^j r_{t+j}, \qquad \Gamma_t = \gamma^{k_t} \prod_{j<k_t} m_{t+j}, \tag{18}$$

and the bootstrapped target is $R_t + \Gamma_t V(s_{t+k})$.

Table 8: Specification of every task and the size of its offline dataset. Observation dimensions are those after the removal described in Section F.1. C is the number of actions in one chunk, and "decisions" is the step limit divided by C. *one of the six tasks used in the main paper; †one of the nine long-horizon tasks.

Task	Obs.	Action/dec.	C	Step limit	Decisions	Trans. (M)	Shards
block-single 1	22	7×24	24	1,250	53	7.3	2
block-double 1	31	7×24	24	2,500	105	53.4	9
block-double 2*	31	7×24	24	2,500	105	45.9	9
block-triple 1	40	7×24	24	3,750	157	286.5	47
block-triple 2	40	7×24	24	3,750	157	273.4	46
block-quadruple 1†	49	7×24	24	5,000	209	403.3	63
block-quadruple 2*†	49	7×24	24	5,000	209	390.1	61
block-quadruple 3†	49	7×24	24	5,000	209	432.3	63
chamber-easy 1	47	7×32	32	4,000	125	19.6	5
chamber-easy 2	47	7×32	32	4,000	125	32.9	5
chamber-easy 3	47	7×32	32	4,000	125	33.7	5
chamber-medium 1*†	56	7×32	32	5,000	157	128.9	18
chamber-medium 2*†	56	7×32	32	5,000	157	134.3	18
chamber-hard 1*†	65	7×32	32	5,500	172	488.6	63
chamber-hard 2*†	65	7×32	32	5,500	172	489.9	65
hanoi-single 1	33	7×32	32	1,000	32	7.1	3
hanoi-single 2	33	7×32	32	1,000	32	7.2	3
hanoi-double 1	38	7×32	32	3,000	94	78.5	24
hanoi-double 2	38	7×32	32	3,000	94	79.5	24
hanoi-triple 1†	43	7×32	32	7,000	219	592.2	181
hanoi-triple 2†	43	7×32	32	7,000	219	576.9	176

Restriction to successful episodes. On each task, every method uses the same choice as the shared proposal policy of Section F.5: successful episodes only on every task except chamber-medium task 1, where the proposal policy, and therefore flow behaviour cloning and our own method, trained on all episodes.

Preprocessing. Applied identically to every method: the six joint-velocity dimensions removed; actions normalised per dimension over the 1st-to-99th percentile range and clipped at ± 5; no frame stacking; no data augmentation; action chunk length C as above.

F.3 Tasks

We use 21 single-task, reward-based manipulation tasks from three families. **block** (single, double, triple, quadruple) presents one to four coloured cubes, to be placed on translucent target markers or stacked into a single tower anywhere on the table. **chamber** (easy, medium, hard) presents a drawer, a sliding window and two lock buttons, one per mechanism, together with one to three cubes; a lock must be released before its mechanism will move and re-engaged before the task counts as complete. Easy tasks involve only the drawer and window, while medium and hard tasks add two or three cubes, which one variant requires inside the closed drawer and the other requires stacked on the table. **hanoi** (single, double, triple) presents three pegs and a stack of one to three holed disks to be moved to another peg, never placing a larger disk on a smaller one.

Each environment ships several task variants, numbered $1, 2, 3$. Variants of one environment share the scene and the distribution of initial states and differ only in the condition that counts as success.

F.4 Training and evaluation settings

Shared proposal policy. One policy per task generates the candidates that every method ranks, and is also the flow behaviour cloning row when evaluated by taking a single sample. It is a residual multilayer perceptron of 4 blocks of width 4096 with layer normalisation and GELU activations (136M parameters), trained by behaviour cloning with a flow-matching objective only and receiving

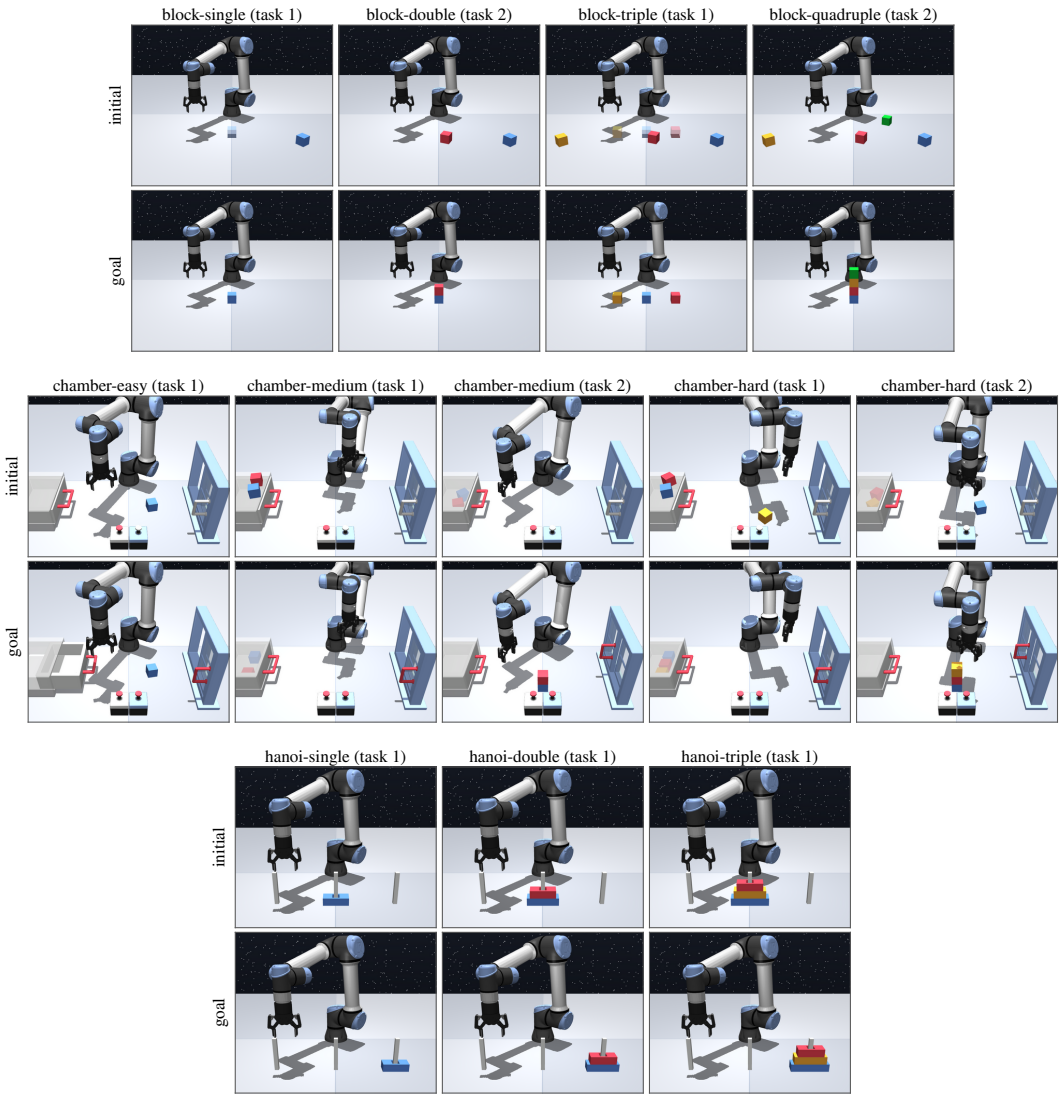


Figure 8: Twelve of the 21 tasks, one row per family (block, chamber, hanoi), including all six tasks used in the main paper. Each panel pair shows the initial state under a fixed reset seed (top) and a goal arrangement (bottom). For chamber-medium and chamber-hard, variant 1 requires the cubes inside the closed drawer and variant 2 requires them stacked. The benchmark supplies no goal image, so each goal frame is constructed from the task's success condition: the task-relevant objects are moved into a satisfying arrangement, the environment's internal post-step update is run, and the environment's own success flag is checked before rendering. The arm is left where the reset placed it.

no gradient from any critic, with flow times drawn from $\mathrm{Beta}(1, 1.5)$ and 10 Euler integration steps at deployment. The checkpoint used ranges from 400,000 to 1,000,000 gradient steps depending on the task.

The number of continuation steps in our estimator is random and capped at 64 (100 when $\gamma = 0.999$); the cap is a safeguard against an unbounded loop and is not tuned.

Early stopping. Let m_j be the success rate at the j-th evaluated checkpoint, averaged over the three training seeds, and let $b_j = \max_{i \le j} m_i$. A checkpoint counts as non-improving if it fails to beat the best so far by more than $\delta = 0.02$, and training stops after $P = 2$ consecutive non-improving

Table 9: Success condition for each task used in the main paper.

Task	Success condition
block-double 2	both cubes stacked, anywhere on the table
block-quadruple 2	all four cubes stacked, anywhere on the table
chamber-hard 1	all cubes inside the closed drawer, both locks re-engaged
chamber-hard 2	all cubes stacked on the table, both locks re-engaged
chamber-medium 1	all cubes inside the closed drawer, both locks re-engaged
chamber-medium 2	all cubes stacked on the table, both locks re-engaged

Table 10: Settings held identical for every method and every task.

Gradient steps $\times$ batch size	$10^6 \times 1024$, i.e. 1.024×10^9 training examples
Checkpoints evaluated	$\{0.2, 0.4, 0.6, 0.8, 1.0\} \times 10^6$ gradient steps
Data residency	3 shards in memory, replaced every 10^4 gradient steps
Gradient clipping	none
Training seeds	3; seed k of a scorer is paired with seed k of the proposal policy
Candidate actions per decision	32 (1 for flow behaviour cloning, which performs no selection)
Evaluation episodes	50 per training seed
Reporting episodes	a fixed set of 50 initial states, identical for every method, seed and checkpoint
Tuning episodes	a disjoint fixed set of 50 initial states
Evaluation determinism	autotuning and command buffers disabled; memory preallocation disabled
Hardware	one NVIDIA H100 80GB per run; the smaller world models share a GPU two at a time

checkpoints:

$$u_j = \begin{cases} 0 & \text{if } m_j > b_{j-1} + \delta, \\ u_{j-1} + 1 & \text{otherwise,} \end{cases} \qquad j^\star = \min\{j : u_j \geq P\}. \tag{19}$$

The same δ and P apply to every method and task; the earliest possible stop is after the 600,000-step checkpoint. Both the value at the stopping point and the best value up to it are reported.

F.5 Methods

Every method differs in one respect only: how it scores a candidate action. For each task a single shared proposal policy π generates $n = 32$ candidate action chunks at every decision; each method scores them with its own estimate $\hat{Q}(s, a)$ and executes the highest-scoring one. Where a method imagines further actions, those actions are drawn from the same shared policy, and where it reads a learned reward it is the same reward predictor. Flow behaviour cloning is evaluated at $n = 1$, executing a single sample of the proposal policy with no selection.

Flow behaviour cloning. Rectified-flow behaviour cloning over the whole action chunk: with x_1 the chunk of actions from the data, $x_0 \sim \mathcal{N}(0, I)$, flow time $t \sim \mathrm{Beta}(1, 1.5)$ and $x_t = (1 - t)x_0 + tx_1$, the objective is $\mathbb{E}\|v_\theta(s, x_t, t) - (x_1 - x_0)\|^2$. At deployment the learned velocity field is integrated once from noise. No critic and no selection among candidates.

Implicit flow Q-learning (IFQL). The flow-based counterpart of implicit diffusion Q-learning: implicit Q-learning for the value, a flow behaviour cloning actor, and deployment by scoring sampled actions and keeping the best. With expectile τ and the multi-step reward and discount of Section F.2,

$$\mathcal{L}_V = \mathbb{E}\big[|\tau - \mathbb{1}\{Q - V < 0\}|\,(Q - V)^2\big], \qquad \mathcal{L}_Q = \mathbb{E}\big[(R_t + \Gamma_t V(s_{t+k}) - Q(s, a))^2\big], \tag{20}$$

both realised as a cross-entropy over a discretised value distribution rather than a squared error. The critic is a two-member ensemble over a discretised value distribution on $[0, 1]$, a residual multilayer perceptron of width 512 and 4 layers (4.6M), with a separate value network (2.2M); expectile 0.5; bootstrapping 200 environment steps ahead; $\gamma = 0.9999$ per environment step; target-network rate 10^{-3}. It scores the shared proposal policy's candidates with the ensemble-mean critic, and its own

Table 11: Training hyperparameters. Values apply to all tasks unless a task is named. "MLP" abbreviates multilayer perceptron.

	Ours	IFQL	LeWM	Diffusion Forcing
Optimiser	Adam	Adam	AdamW, weight decay 10^{-3}	Adam
Learning rate	10^{-4}	10^{-4}	5×10^{-5} main, 10^{-4} other	10^{-4}
Main network	MLP width 1024, 3 layers	critic: residual MLP, width 512, 4 layers	encoder 512, 3 layers; transformer 6 layers; 16 heads, embed. 192	3 layers, width 672, embed. 128
Training horizon	$\beta \sim \mathcal{U}(0.93, 0.995)$, drawn per example	γ=0.9999, bootstrap 200 steps	—	windows of 8 tokens
Sampling steps	8 Euler	—	—	8 diffusion
Reward predictor	sigmoid MLP 512, 2 layers, weighted cross-entropy	none: trains on the environment reward	trained, unused at evaluation	—
Loss weights	future-state 5	expectile 0.5	embedding regulariser 0.09	—

Table 12: Size of the module that computes an action's score, excluding target-network and moving-average copies. The 136M proposal policy is shared by every method and excluded.

Method	Scoring modules	Parameters
Ours	future-state model + reward predictor	6.1M + 0.36M
Diffusion Forcing	backbone + initial-state head	5.8M
IFQL	critic ensemble + value network	6.7M
LeWM	encoder + predictor + decoder	13.6M

actor is unused at evaluation. It trains on the environment's sparse reward, and its batch size, training data and critic width match ours on every task.

World-model baselines and the shared planner. Two baselines learn a dynamics model over action chunks, so that one model prediction advances the environment by C steps, and both are scored with the same planner. From a candidate first chunk a_0, the planner imagines M rollouts of H predictions, drawing each subsequent action from the shared proposal policy and reading the shared reward predictor at each imagined state:

$$\hat{Q}(s, a_0) = \frac{1}{M} \sum_{m=1}^{M} \sum_{h=1}^{H} \gamma_c^{h-1} \hat{r}\big(\hat{s}_h^{(m)}\big), \qquad \hat{s}_h^{(m)} \sim \hat{P}\big(\cdot \mid \hat{s}_{h-1}^{(m)}, a_{h-1}^{(m)}\big), \quad \hat{s}_0 = s, \tag{21}$$

with $\gamma_c = 0.99$ per prediction. ***LeWM*** is a latent joint-embedding world model, trained with a joint-embedding prediction objective, a covariance-style regulariser on the embeddings, and a decoder back to observations so that the shared reward predictor can be applied; encoder: multilayer perceptron of width 512 and 3 layers; predictor: 6-layer transformer, 16 heads, embedding dimension 192, feed-forward width 2048; 13.6M parameters. ***Diffusion Forcing*** is a causal sequence model over windows of 8 tokens, each a state and action-chunk pair, trained with an independent noise level per token; 3 layers of width 672, embedding dimension 128, 8 diffusion steps at sampling, 5.8M parameters, with goal-satisfying states treated as absorbing.

Our method. We learn a discounted successor measure $d_\beta^\pi(\cdot \mid s, a)$, a distribution over the states visited in the future discounted at rate β, by conditional flow matching. For a state-action pair (s_t, a_t) from the data we draw a horizon $\beta \sim \mathcal{U}(0.93, 0.995)$ and a lookahead $k \sim \mathrm{Geom}(1-\beta)$ and regress the flow onto the future state $s_f = s_{t+k}$, clipped at the end of the episode and represented as a normalised difference from s_t. A reward predictor $\hat{r}(s_f) \in [0, 1]$, a sigmoid-output network of width 512 and 2 layers, is trained with class-imbalance-weighted binary cross-entropy on the

Table 13: Settings chosen at evaluation time rather than during training. Where a setting is chosen per task, the reported number is an argmax over the grid and is labelled as such, alongside the best setting held fixed across all tasks. A discount of $\gamma = 0.99$ per environment step was chosen on every task.

Method	Grid searched	Chosen
Ours	$\gamma \in \{0.99, 0.995, 0.999\}$; horizon $\beta \in \{0.93, 0.95, 0.97, 0.98958, 0.995\}$ subject to $\beta \leq \gamma$; continuation chains $m \in \{8, 32, 64\}$, with $m > 8$ only at γ=0.99	on the tuning episodes, then re-measured on the reporting episodes
LeWM planner	lookahead $H \in \{2, 5, 10, 20\}$ predictions $\times$ rollouts $M \in \{8, 32\}$	per task
Diffusion Forcing planner	$H \in \{5, 10, 20, 40, 80\}$, $M = 8$	per task
IFQL, flow behaviour cloning	none	—

success indicator; its target is the reward at that single state and not the discounted return from it. At evaluation, each of m independent chains draws a future state $s_1 \sim d^\pi_\beta(\cdot \mid s, a)$, then continues with probability

$$p = \frac{\gamma - \beta}{\gamma(1 - \beta)} \qquad (\beta \leq \gamma \text{ required}), \tag{22}$$

drawing a fresh action $a_\ell \sim \pi(\cdot \mid s_\ell)$ and a fresh future state $s_{\ell+1} \sim d^\pi_\beta(\cdot \mid s_\ell, a_\ell)$ each time, and otherwise stops at a state s_L. The reward is read only at the state where the chain stopped:

$$\hat{Q}^\pi(s, a) = \frac{1}{(1 - \gamma)m} \sum_{i=1}^{m} \hat{r}\big(s^{(i)}_{L_i}\big). \tag{23}$$

The total lookahead at the stopping point is geometrically distributed with parameter $1 - \gamma$, so the stopping state follows $d^\pi_\gamma(\cdot \mid s, a)$ and the estimate is unbiased for $Q^\pi(s, a)$ for every $\beta \leq \gamma$. The values of β, γ, and m are set at evaluation time (Table 13).

Methods not included. We exclude a sampling-based planner for LeWM, whose three seeds were scored with two different versions of the action-clipping code; and flow Q-learning, which deploys a single action from a distilled one-step policy and performs no selection among candidates. Deterministic behaviour cloning is retained only as a sanity floor.

G Experimental Details for Test-Time Compute Scaling

This appendix specifies the evaluation protocol, the models, and every hyperparameter behind the test-time compute experiments: scaling with the number of action samples, the recursion depth of RSM, the matched planning-horizon comparison, and the success–inference-time frontier.

All experiments use the hardest chamber task of OCBench (task 1), in which the agent must place three cubes inside a drawer; episodes last at most 5,500 environment steps. We follow the same evaluation protocol in the rest of OCBench experiments.

G.1 Configurations per Experiment

Number of action samples. Each method uses a single configuration at every N, chosen as the one with the highest mean success over $N \in \{1, \ldots, 64\}$, so the error bars reflect seed variation rather than per-N selection: RSM with $\beta = 0.98958$, $m = 8$, $E = 8$; LeWM with $H = 128$, $K = 1$; DFWM with $H = 160$, $M = 8$; and IFQL. We evaluate $N \in \{1, 2, 4, 8, 16, 32, 64, 128, 256\}$; IFQL is additionally evaluated up to $N = 4096$.

Recursion depth. RSM uses $N = 32$, number of chains of recursion $m = 8$, number of flow model euler integration steps $E = 8$, and $\gamma = 0.99$, with $\beta \in$

$\{0.93, 0.95, 0.97, 0.98, 0.982, 0.985, 0.9875, 0.98958, 0.99\}$, which spans 1 to 7.4 expected hops per chain. The reference lines are the best LeWM and DFWM configurations at the same $N = 32$.

Matched planning horizon. All methods use $N = 16$. The world models roll out H environment steps (LeWM with $K = 1$, DFWM with $M = 8$). RSM targets the same horizon by setting $\gamma = 1 - 1/H$, so that its effective horizon $1/(1-\gamma)$ equals H; for each H we evaluate the RSM configurations listed in Table 14 and report the best. The world models use a single configuration per H. For LeWM, the per-chunk discount of 0.725 limits the effective horizon to about 116 steps regardless of rollout length, so beyond 128 steps its horizon is matched in rollout length only.

Table 14: Matched planning horizon ($N = 16$). RSM configurations evaluated at each horizon (three seeds each), the selected configuration, and the success rate (%) of each method.

H	RSM γ	# configs	Selected RSM configuration	RSM	LeWM	DFWM
32	0.96875	2	$\beta = 0.96875, m = 4$	75.3 ± 2.9	58.0 ± 3.5	54.0 ± 5.0
64	0.98438	4	$\beta = 0.98, m = 4$	80.0 ± 2.0	58.0 ± 3.5	59.3 ± 3.5
128	0.99219	5	$\beta = 0.99, m = 4$	77.3 ± 1.8	60.0 ± 2.1	72.0 ± 3.5
256	0.99609	13	$\beta = 0.985, m = 32$	67.3 ± 3.3	47.3 ± 1.8	59.3 ± 9.0

The RSM search spaces were: $H = 32$: $\beta \in \{0.95, 0.96875\}$, $m \in \{4\}$; $H = 64$: $\beta \in \{0.95, 0.97, 0.98, 0.984375\}$, $m \in \{4\}$; $H = 128$: $\beta \in \{0.95, 0.97, 0.98, 0.99, 0.992188\}$, $m \in \{4\}$; $H = 256$: $\beta \in \{0.97, 0.98, 0.9825, 0.985, 0.9875, 0.99, 0.995\}$, $m \in \{4, 8, 16, 32\}$, $E \in \{8, 16\}$. Values of β are restricted to $\beta \leq \gamma$ and to the range seen in training.

More action samples at a 256-step horizon. Figure 9 repeats the 256-step comparison with $N \in \{16, 32, 64\}$. For RSM we report the best configuration at each N; the world models use the configurations above.

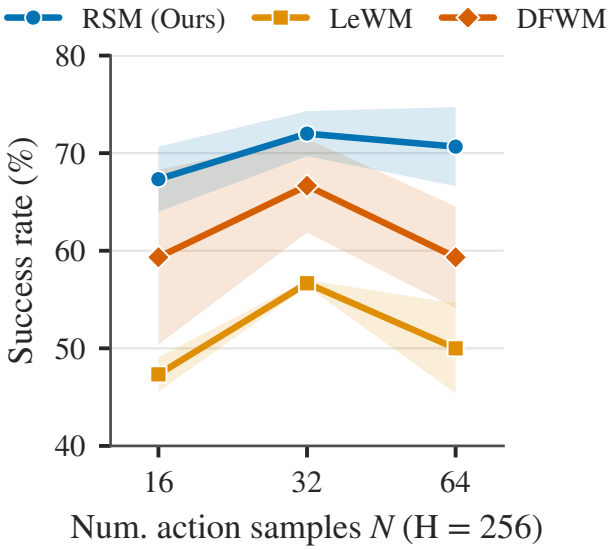


Figure 9: Success rate at a matched 256-step planning horizon for $N \in \{16, 32, 64\}$ (mean $\pm$ standard error over three seeds). RSM: $N = 16$: 67.3%, $N = 32$: 72.0%, $N = 64$: 70.7%.

Inference-time frontier. For each method we evaluate the configurations in Table 15 and plot, for every inference-time budget, the best success rate reached within that budget. Inference time per decision is $(N + c) \times t_\pi$, where t_π is the time of one sequential sample from the proposal policy and c is the scoring cost measured in the same unit. All methods are timed in a single process on one dedicated GPU, over four replicates (median relative spread 3%), and reported with $t_\pi = 8.05$ ms;

expressing cost in units of t_π removes hardware differences. Configurations timed in separate runs are placed on a common scale using configurations shared between runs (rescaling factors 0.983 and 0.992).

Table 15: Configurations evaluated for the inference-time frontier. E: Euler steps per hop; m: chains; H: horizon in environment steps; K, M: rollouts.

Method	Hyperparameter values	# configs	N
RSM	$\beta \in \{0.93, 0.95, 0.97, 0.98, 0.982, 0.985, 0.9875, 0.98958, 0.99\}$; $m \in \{1, 2, 4, 8, 16\}$; $E \in \{1, 2, 4, 8, 16, 32\}$	38	$\{1, 2, 3, 4, 8, 16, 32, 64\}$
LeWM	$H \in \{32, 64, 96, 128, 160, 256, 320, 512, 640\}$; $K \in \{1, 4, 8, 16, 32\}$	21	$\{1, 2, 3, 4, 8, 16, 32, 64\}$
DFWM	$H \in \{32, 96, 160, 256, 320, 480, 640\}$; $M \in \{1, 8\}$	10	$\{1, 2, 3, 4, 8, 16, 32, 64, 128, 256\}$
IFQL	no test-time parameters	1	$\{1, 2, 3, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096\}$

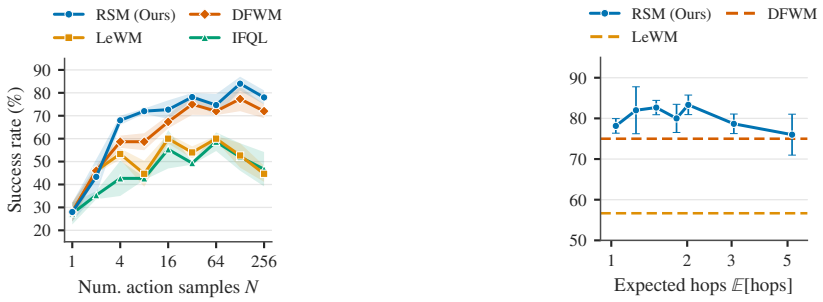
G.2 Additional Experimental Results

Table 16: RSM achieves the best success rate and throughput across long-horizon OCBench tasks, outperforming value-based and world-model planning baselines. Metric definitions and evaluation details are provided in Section 5.1.

Task	BC	TD-Flow	IFQL	LeWM	DFWM	RSM (ours)
block-quadruple 1	18 ± 3 (0.19)	1 ± 1 (0.01)	37 ± 1 (0.41)	21 ± 5 (0.22)	71 ± 7 (1.01)	**87** ± 2 (**1.36**)
block-quadruple 2	11 ± 3 (0.11)	5 ± 1 (0.05)	12 ± 4 (0.13)	18 ± 1 (0.19)	63 ± 3 (0.88)	**73** ± 7 (**1.04**)
block-quadruple 3	41 ± 5 (0.46)	24 ± 3 (0.25)	61 ± 7 (0.76)	52 ±11 (0.63)	75 ± 6 (1.06)	**82** ± 6 (**1.13**)
chamber-hard 1	23 ± 5 (0.24)	8 ± 2 (0.08)	41 ± 1 (0.46)	52 ± 1 (0.60)	75 ± 5 (0.96)	**79** ± 1 (**1.06**)
chamber-hard 2	14 ± 2 (0.15)	5 ± 2 (0.05)	14 ± 2 (0.15)	23 ± 6 (0.25)	31 ± 3 (0.35)	**41** ± 2 (**0.48**)
chamber-medium 1	31 ± 6 (0.33)	11 ± 5 (0.12)	83 ± 2 (1.19)	71 ± 2 (0.92)	79 ± 2 (1.03)	**87** ± 5 (**1.21**)
chamber-medium 2	52 ± 6 (0.60)	33 ± 4 (0.36)	45 ± 1 (0.53)	55 ± 8 (0.68)	68 ± 5 (0.91)	**72** ± 3 (**0.95**)
hanoi-triple 1	61 ± 8 (0.71)	44 ± 9 (0.51)	81 ± 4 (1.18)	87 ± 4 (1.27)	89 ± 5 (1.27)	**93** ± 2 (**1.28**)
hanoi-triple 2	52 ±14 (0.65)	30 ±11 (0.38)	45 ± 9 (0.58)	49 ± 9 (0.70)	60 ±15 (0.82)	**77** ±11 (**0.98**)
Average	34 (0.38)	18 (0.20)	47 (0.60)	47 (0.61)	68 (0.92)	**77** (**1.06**)

* Success rate ± s.e. (throughput).

G.2.1 Test-Time Compute Analysis



(a) Scaling with the number of action candidates.

(b) Sensitivity to recursion depth.

Figure 10: **Test-time compute analysis. (a)** RSM and DFWM benefit from increasing the number of sampled action candidates. **(b)** RSM has an intermediate optimal recursion depth while remaining robust across a broad range of β.

Action-candidate scaling. Figure 10a varies the number of actions N proposed at each decision while keeping each method's scoring procedure fixed. RSM improves steadily as the candidate set

grows, showing that its successor-based value estimates can reliably rank increasingly diverse proposals. DFWM also benefits from additional candidates, but LeWM and IFQL improve much less, suggesting that compounding world-model error and TD value-estimation error limit the usefulness of extra test-time samples when the ranking signal is inaccurate.

Recursion-depth sensitivity. Figure 10b varies the successor discount β, which controls the expected number of recursive hops used to construct the target long-horizon distribution. Performance is highest at an intermediate depth: too little recursion limits long-horizon composition, whereas excessive recursion compounds modeling error across hops. RSM nevertheless remains above the non-recursive baselines throughout the sweep, indicating that the method is robust to a reasonably broad range of β while still benefiting from a moderate amount of recursive composition.

H Subgoal Generation Experiment Details

Tasks and Datasets. We use standard tasks from OGBench (Park et al., 2025a) with sparse reward.

Baselines:
IQL: Implicit Q-Learning. Policy extracted with DDPG+BC. Represents flat single-step TD-based offline RL.
IFQL: Flow BC with action selection based on an IQL Q-function.
SWM: Single-step World Model. Plan with an FBC prior until reaching the goal. Represents planning with a single-step world model.
SWM+V: Single-step World Model that uses and IQL Value function for trajectory scoring. Plan with an FBC prior for a fixed horizon and select the plan based on the final state's value. Represents subgoal planning with a single-step world model.
HIQL: based on Hierarchical Implicit Q-Learning (Park et al., 2023). Low-level policy reuses the IQL policy, while the high-level policy is extracted with AWR. Goal representations are disabled.
SHARSA: based on hierarchical SARSA (Park et al., 2026b). Flow BC + value-based selection on both levels. The high level n-step value is used for subgoal selection while the low level is either IQL or IFQL. **TD-Flow**: identical to RSM except for the training objective which uses TD.

RSM, SHARSA and TD-Flow were evaluated with both IQL and IFQL, best results are reported.

Table 17: **OGBench Subgoal Generation.** Results are averaged over 3 seeds. Best results up to a standard deviation are highlighted in bold.

Environment	Dataset	IQL	IFQL	SWM	SWM+V	HIQL	SHARSA	TD-RSM	RSM
pointmaze	medium-navigate-v0	89 ±3	93 ±1	**97** ±2	**99** ±2	90 ±2	**100** ±1	**99** ±1	**99** ±1
	large-navigate-v0	61 ±5	70 ±8	88 ±6	87 ±4	**98** ±2	**99** ±0	**100** ±1	**99** ±1
	medium-stitch-v0	53 ±11	77 ±4	67 ±6	**99** ±1	87 ±4	94 ±1	**97** ±2	**99** ±1
	large-stitch-v0	18 ±5	20 ±14	0 ±1	40 ±1	32 ±5	67 ±5	49 ±13	**83** ±8
antmaze	medium-navigate-v0	65 ±10	88 ±2	67 ±5	59 ±2	**93** ±5	**95** ±5	65 ±7	**94** ±2
	large-navigate-v0	51 ±7	**74** ±2	20 ±2	25 ±5	61 ±11	**76** ±7	**70** ±7	**75** ±2
	medium-stitch-v0	45 ±2	65 ±5	33 ±6	46 ±1	82 ±4	**95** ±4	55 ±10	82 ±7
	large-stitch-v0	52 ±13	48 ±3	4 ±3	15 ±4	64 ±10	**84** ±3	62 ±11	**79** ±9
manipulation	cube-single-play-v0	**94** ±3	79 ±4	91 ±2	86 ±4	50 ±6	62 ±6	**94** ±2	**93** ±2
	cube-double-play-v0	77 ±6	65 ±8	39 ±5	62 ±6	33 ±1	20 ±3	77 ±2	**83** ±4
	cube-triple-play-v0	8 ±6	23 ±2	12 ±4	21 ±6	6 ±4	7 ±3	**25** ±6	**26** ±2
	scene-play-v0	54 ±8	52 ±2	**73** ±8	63 ±3	36 ±4	22 ±5	60 ±7	**70** ±5
visual-antmaze	medium-navigate-v0	**92** ±3	62 ±3	—	—	**90** ±5	**90** ±5	**92** ±3	**93** ±3
	medium-stitch-v0	73 ±2	16 ±4	—	—	79 ±3	**90** ±4	68 ±11	**89** ±3
	large-navigate-v0	66 ±6	47 ±5	—	—	**71** ±7	**76** ±13	64 ±11	**77** ±12
visual-manip	cube-single-play-v0	42 ±7	54 ±2	—	—	19 ±12	**57** ±17	48 ±9	**62** ±6
	cube-double-play-v0	5 ±4	16 ±4	—	—	1 ±1	**21** ±1	15 ±4	**18** ±4
	scene-play-v0	14 ±1	48 ±3	—	—	13 ±3	**53** ±9	**51** ±6	**56** ±4

I Zero-Shot Reinforcement Learning

I.1 Extending RSM to Zero-Shot RL

Zero-Shot Reinforcement Learning (Borsa et al., 2019; Touati & Ollivier, 2021; Touati et al., 2023) is an emerging RL paradigm where the learner is given an offline transition dataset at training time

Algorithm 2 RSM Subgoal Planning

Require: state s, goal g; occupancy model d_β; goal classifier c; number of particles N; reachers n_{reach}; max hops H
Ensure: subgoal s_{sub}
$s_0^i \leftarrow s$ for $i = 1, \ldots, N$
for $t = 1, \ldots, H$ **do**
  $s_t^i \sim d_\beta(\cdot \mid s_{t-1}^i, g)$ for all particles that have not reached g ▷ reached particles stay at g
  $h^i \leftarrow t$ for particles that newly satisfy $c(s_t^i, g) = 1$ ▷ arrival hop
  break if at least n_{reach} particles have reached g
end for
if no particle reached g **then**
  return g
end if
$\tau^\star \leftarrow$ median of the distinct arrival hops $\{h^i\}$
return an intermediate state $s_k^{i^\star}$ of the selected trajectory, $1 \le k < \tau^\star$

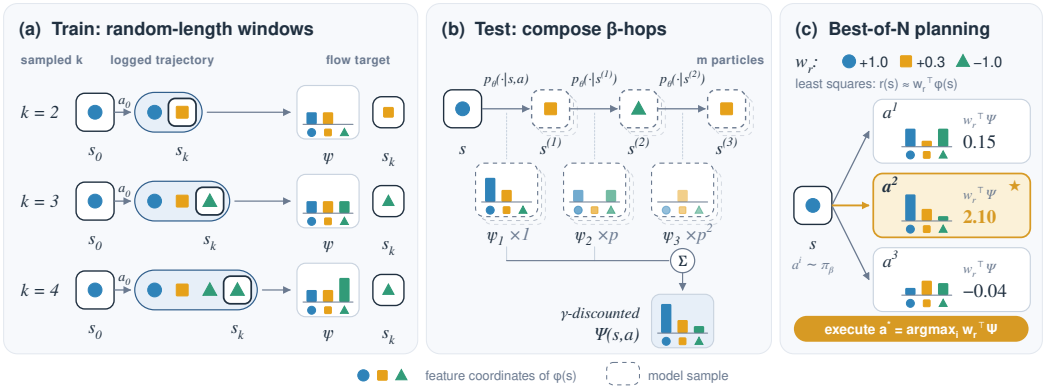


Figure 11: **RSM for Zero-Shot RL:** RSM can be extended to zero-shot RL by generating both the future state and the sum of features in between. By averaging over multi-hop particles, we can sample from the successor feature distribution of the behavior policy. Linear-regression recovers the downstream reward weight, which enables best-of-n planning.

and needs to adapt to arbitrary downstream reward at test time. To extend RSM to zero-shot RL, we build upon the notion of successor features (Barreto et al., 2018): similar to value functions which estimate an expectation over discounted sum of rewards, successor features estimate an expectation over discounted sum of state features. Suppose that the downstream reward is linear in the state feature, the value can be recovered immediately via linear regression. Given an offline dataset, we can estimate a *distribution* of successor features conditioned on current state and immediate action for the behavior policy, then plan at test time by selecting the action that leads to the best value Zhu et al. (2024). Following this principle, we extend RSM to RSM-ZS. At training time, we train a flow model $p_\theta(\psi, s_k, a_0 \mid s_0)$, where s_k is a future state at a geometric offset and ψ is the *undiscounted* sum of features up to it:

$$k \sim \text{Geom}(1-\beta),\ k \in \{0, 1, \ldots\}, \qquad \psi = \textstyle\sum_{j=1}^{k} \phi(s_j). \tag{24}$$

Step j enters the sum iff $k \ge j$, which happens with probability β^j. Hence $\mathbb{E}_k[\psi] = \sum_{j \ge 1} \beta^j \phi(s_j)$, and ψ is a Monte Carlo sample of the β-discounted successor feature. The discount lives in the stopping time rather than the target, which is what lets hops compose. We fit $x = (\psi, s_k, a_0)$ with Eq. equation 6, conditioned on s_0, and on a_0 only with probability p_a (masked otherwise). One network therefore gives both an action-conditioned model $p_\theta(\psi, s_k \mid s_0, a_0)$ and an unconditional model whose action marginal is π_β.

Multi-hop composition. To reach a horizon $\gamma > \beta$, we chain β-hops as in Section 4.2: $(\psi_1, s^{(1)}) \sim p_\theta(\cdot \mid s, a)$, then $(\psi_h, s^{(h)}) \sim p_\theta(\cdot \mid s^{(h-1)})$ for $h \ge 2$, with actions drawn from

π_β. Continuing with probability p makes the total offset $\mathrm{Geom}(1-\gamma)$, so the chained sum has expectation $\sum_{j\geq 1}\gamma^j\phi(s_j)$. Rather than sampling the hop count $L\sim\mathrm{Geom}(1-p)$, we integrate it out using $P(L\geq h)=p^{h-1}$, and we average m particles:

$$\Psi(s,a)=\frac{1}{m}\sum_{\ell=1}^{m}\sum_{h=1}^{H}p^{h-1}\,\psi_h^{(\ell)}, \tag{25}$$

truncated at H hops (discarded mass p^H). This removes hop-count variance and fixes compute. A single model trained at β serves any $\gamma>\beta$ at test time.

Best-of-N planning. Given reward-labeled states, we fit $w_r=\arg\min_w\sum_s(r(s)-w^\top\phi(s))^2+\lambda\|w\|^2$. At each step, we draw N candidate actions a^i from the unconditional model at s and execute

$$a^\star=\arg\max_{i\leq N}\; w_r^\top\Psi(s,a^i). \tag{26}$$

I.2 Zero-shot Experiment Details

Environments and Baselines We evaluate the zero-shot RL method, RSM-ZS on two benchmarks: ExoRL (Denis Yarats, 2022) and OGBench (Park et al., 2025a). Each benchmark contains a number of tasks suites, where each suite provides an offline transition dataset and a number of downstream tasks to transfer to. We use the same evaluation protocol as (Zheng et al., 2026) and compare to the following baselines. Laplacian (Wu et al., 2019), BYOL-γ (Lawson et al., 2026), ICVF (Ghosh et al., 2023), HILP (Park et al., 2024) follow the Universal Successor Feature (Borsa et al., 2019) framework, where a separately learned feature $\phi(s)$ is used to train a random-z-conditioned actor-critic on the reward $\phi(s)^\top z$ with a scalar $Q(s,a,z)$, and a test-time z is recovered via linear regression and used to query the model. TD-JEPA (Bagatella et al., 2026b), FB (Touati & Ollivier, 2021) and One-Step FB (Zheng et al., 2026) are zero-shot RL methods that jointly learn the features with successor measure objectives.

Hyperparameters We use a near-uniform set of hyperparmeters for RSM-ZS for each benchmark. The flow model is a 4-layer MLP with 2048 hidden units each. For ExORL, we use a training-time discount $\beta=0.2$, test-time $\gamma=0.95$, feature estimation batch size $m=4$, and planning batch size $n=32$. On OGBench, we use $\beta=0.96$, $\gamma=0.99$, $m=16$, $n=4$. We use with identity features for locomotion tasks and concatenated identity and HILP features for manipulation tasks.

Table 18: **Zero-shot RL results.** mean $\pm$ sample standard deviation over 3 seeds of 50 rollouts.

Domain	Laplacian	BYOL-γ	TD-JEPA	ICVF	HILP	FB	One-Step FB	RSM-ZS (ours)
walker (4 tasks)	228 ± 2	227 ± 2	316 ± 12	$\mathbf{619\pm23}$	393 ± 108	400 ± 40	379 ± 26	474 ± 12
cheetah (4 tasks)	125 ± 41	127 ± 39	185 ± 7	187 ± 13	116 ± 78	271 ± 46	378 ± 56	$\mathbf{554\pm9}$
quadruped (4 tasks)	462 ± 35	496 ± 35	461 ± 115	546 ± 37	352 ± 59	246 ± 31	645 ± 15	$\mathbf{733\pm8}$
jaco (4 tasks)	3 ± 1	3 ± 0	13 ± 4	23 ± 3	20 ± 5	10 ± 4	22 ± 4	$\mathbf{66\pm14}$
ExORL mean	204.5	213.3	243.8	343.8	220.3	231.8	356.0	**456.8**
antmaze-large-navigate (5 tasks)	9 ± 1	21 ± 2	14 ± 9	23 ± 3	$\mathbf{34\pm2}$	25 ± 5	30 ± 9	25 ± 3
antmaze-teleport-navigate (5 tasks)	3 ± 1	16 ± 4	4 ± 1	$\mathbf{29\pm3}$	19 ± 6	16 ± 8	11 ± 6	18 ± 4
cube-single-play (5 tasks)	6 ± 2	13 ± 2	8 ± 4	13 ± 2	30 ± 8	2 ± 1	3 ± 2	$\mathbf{66\pm1}$
scene-play (5 tasks)	4 ± 1	15 ± 8	6 ± 1	8 ± 6	$\mathbf{19\pm6}$	6 ± 4	8 ± 2	$\mathbf{23\pm2}$
OGBench mean	5.5	16.3	8.0	18.3	25.5	12.3	13.0	**32.6**

J Additional experimental results

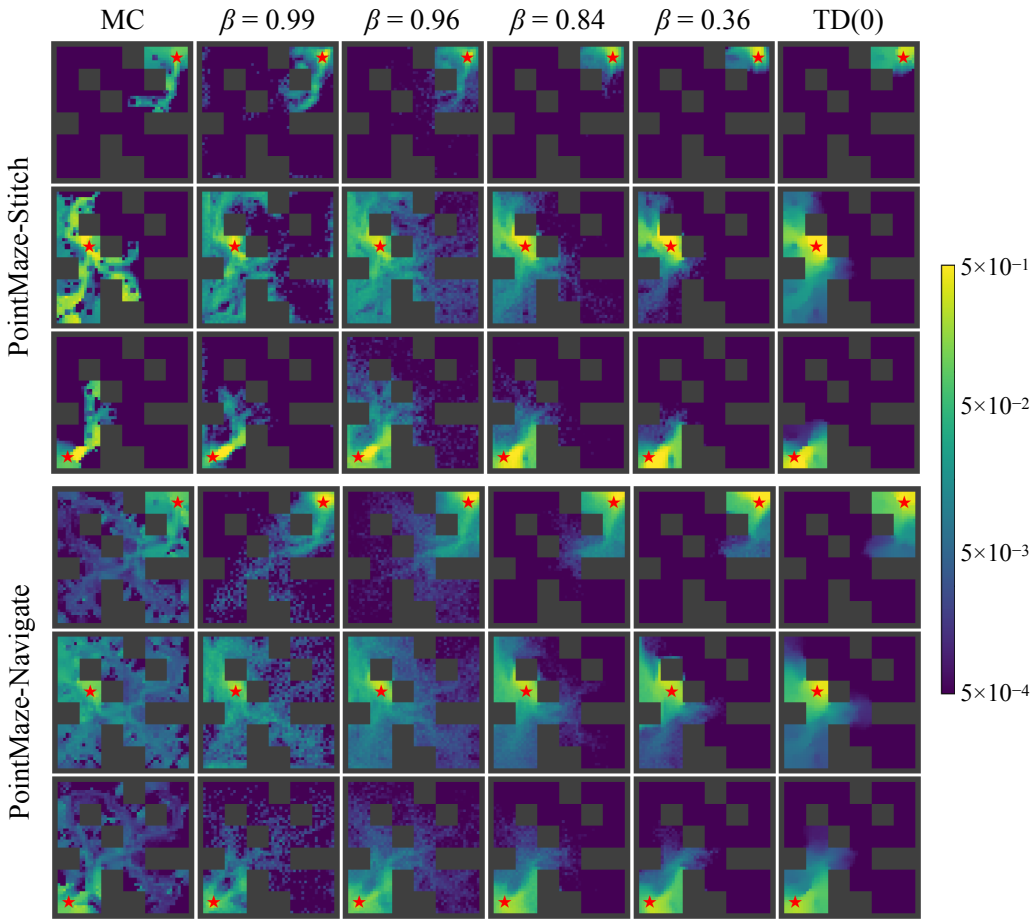


Figure 12: Value functions for reaching the goal (reward is Gaussian distance from red star) learned from Monte-Carlo estimation, TD(0) tabular solve, and RSM at different β values. At low β, RSM resembles a TD-like solution whereas as high β, it resembles an MC-like solution. PointMaze-stitch is an intentionally disconnected dataset while PointMaze-navigate has sprawling goal-reaching demonstrations. Methods are learned action-free and unconditional.