REINFORCED PLANNING WITH LATENT WORLD MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Humans solve complex problems by constructing plans and mentally simulating their outcomes with an internal model of the world. Machine learning has made substantial progress in learning world models that predict the consequences of action sequences, yet the procedures used to plan with these models remain largely hand-designed. Most planners rely on fixed search or optimization rules; approaches that learn aspects of search typically imitate a predefined optimizer or use planning to train an amortized policy, rather than learning how to iteratively improve a complete multi-step plan. We introduce Reinforced Planning, a method for learning the plan-update rule itself through a pretrained world model. Our implementation, RP1, learns a critic for imagined outcomes by temporal-difference learning, and a neural operator that iteratively revises multi-step action sequences, trained by differentiating the critic's value through the world model. RP1 requires no optimizer demonstrations and is trained without environment interaction, from imagined rollouts of a pretrained differentiable latent world model; environment episodes are used only for checkpoint selection. Across visual navigation, arm reaching, and robotic manipulation on two world-model backbones, RP1 matches or exceeds hand-designed and learned planners, achieving near-perfect success in several settings while using $1{,}000\times$ fewer world-model rollouts than the strongest alternative (CEM) and planning up to $67\times$ faster under concurrent planners inference.

1 INTRODUCTION

Humans are commonly understood to solve complex problems by imagining possible futures and evaluating their consequences. Hippocampal activity can represent prospective trajectories before an action is taken [18; 36], supporting the view that the brain uses a learned cognitive map for internal simulation [49]. Computationally, this separates planning into two components: a *world model* predicts the consequences of hypothetical actions, while a *planner* determines how candidate action sequences are generated, evaluated, and improved.

Machine learning has made substantial progress on the first component. Latent world models now support high-dimensional visual prediction [6; 7; 9], self-supervised predictive representations [22; 3], and planning with pretrained, reward-free models [53; 42; 27]. Yet the planner operating on top of these models is still usually hand-designed. Given a candidate action sequence, the world model can predict its outcome, but it does not specify how that sequence should be changed to produce a better one. Existing systems therefore rely on fixed search rules, often requiring thousands of world-model evaluations per decision and configurations that must be chosen separately for different tasks and models [7; 11; 53; 42; 37].

Recent work shows that planning can emerge within neural networks trained through reinforcement learning [4; 48; 43], and learned optimizers show that update rules can be learned from gradients by unrolling the optimizer and training on the resulting losses [1; 23]. It remains open how to learn an explicit planner on top of a separately pretrained world model: existing model-based approaches retain hand-designed update rules, imitate a conventional optimizer, or optimize only the next action rather than an entire action sequence (Sec. 2).

We introduce the **Reinforced Planning** method and its first implementation, RP1, which learns both how imagined outcomes should be evaluated and how multi-step plans should be improved.

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.

RP1 learns a goal-conditioned quasimetric critic [26; 52] from offline trajectories using temporal-difference learning, then trains a neural planner to repeatedly improve action plans at inference time. At each refinement step, RP1 rolls out the current plan through the world model, evaluates its predicted outcome, and supplies the resulting value and action gradient to a learned update operator. The gradient provides information about the local shape of the objective but does not prescribe the update: the planner learns how to turn that information into a new multi-step plan.

We evaluate RP1 with two pretrained world-model backbones, LeWorldModel and PLDM, across visual navigation (TwoRoom), continuous-control reaching (Reacher), and contact-rich manipulation (OGBench Cube). Across these three domains, RP1 matches or exceeds the strongest existing planners while using only 9 world-model rollouts per decision, compared with $9{,}000$ for the strongest competing method, and reduces planning latency by up to $67\times$ when multiple control loops share one GPU.

2 Background

Fixed or inherited planning rules. Most model-based planners use a hand-designed update rule: CEM in PlaNet and DINO-WM, MPPI in the TD-MPC family, or gradient descent through differentiable world-model rollouts [7; 11; 53; 42; 37]. Universal Planning Networks optimize multi-step action sequences but fix gradient descent as the optimizer [44]. DMPO retains an MPPI update and shift operation and learns modifications to them from online task return [39]. L2O-MPC learns the runtime update, but only by imitating a higher-budget MPPI expert that must be run during training [38]. In all these, the planning rules are either (partly) hand-designed or learned from a hand-designed optimizer. Learned optimizers replace hand-designed rules with networks that map gradients to updates and are trained by unrolling [1; 23]; RP1 applies this principle to action plans evaluated through a world model and a learned cost-to-go.

Amortized model-based control. The Dreamer methods use their world model to train an amortized policy and do not plan through the model at inference time [8]. Diffuser learns a generative model over state–action trajectories and refines them by denoising, rather than rolling out and evaluating candidate plans through a separate world model [17]. The Imagination Based Planner (IBP) and Thinker learn which imagined trajectories to construct or inspect, but the gathered information conditions an amortized policy rather than being used to improve a candidate plan [35; 5].

Iterative next-action optimization. Iterative Amortized Policy Optimization (IAPO) learns an iterative optimizer for the current-state action distribution $\pi(a_t \mid s_t)$ [29]. Even in its model-based variant, future actions in a rollout remain policy outputs rather than jointly optimized variables, so the learned optimization improves one action rather than a multi-step plan.

Objectives for imagined plans. JEPA-based world models typically score imagined outcomes by their Euclidean distance to the goal in latent space [27; 42; 53]. However, latent proximity need not reflect temporal reachability (Theorem 1), and learned reachability objectives can outperform latent distance [24]. Scoring outcomes by value estimates is well established in model-based control [10; 11]. We therefore estimate temporal cost-to-go from experience using a goal-conditioned quasimetric-style critic [19; 40; 2; 12; 26; 52; 51].

Scope of comparison. We compare against planners: methods that roll out candidate plans through the world model at inference time and revise them based on the predicted outcome, including the learned planners DMPO and L2O-MPC. Amortized policies are a different family and outside our scope. A planner checks each decision against the world model before executing it, and RP1 uses the same interface as the planners it replaces, so it can be substituted into existing latent-planning pipelines without changing the world model.

3 Preliminaries

We consider a goal-conditioned MDP $(\mathcal{S}, \mathcal{A}, \mathcal{T}, g, \rho_0)$ with state space $\mathcal{S}$, action space $\mathcal{A} \subset \mathbb{R}^{|a|}$, goal state $g \in \mathcal{S}$, transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ and initial-state distribution ρ_0. The agent does not observe s directly, but instead receives an observation o in the form of a visual image.

World model. A world model predicts the next state, given the current state and some candidate action. Specifically, the neural network maps an observation o_t to its latent representation z_t through an encoder $z_t = E_\phi(o_t) \in \mathcal{Z}$. The world model then predicts the next latent state via a prediction map $h_\phi(z, a) = \hat{z}$, where $\hat{z}$ is the predicted (or *imagined*) latent of the next state. We define the N-step rollout operator of the world model via

$$H_\phi(\mathbf{a}, \hat{z}_0) := h_\phi\big(h_\phi(\cdots h_\phi(\hat{z}_0, a_0)\cdots, a_{N-2}),\, a_{N-1}\big) \;=\; \hat{z}_N, \tag{1}$$

a composition of N forward rolls of the world model.

Model-based Planning. Given a start latent z_t and encoded goal z_g, a plan is scored by a terminal cost C applied to the final latent state, as predicted by the world model H_ϕ. This plan aims to optimize the objective

$$J(\mathbf{a}; z_t, z_g) \;:=\; C\big(H_\phi(\mathbf{a}, z_t),\, z_g\big). \tag{2}$$

A model-based planner maintains a planner state ω_k, which may contain a candidate plan, a population of plans, a sampling distribution, or optimizer state. Given the fixed world model and planning objective, it repeatedly applies an update rule

$$F : \omega_k \mapsto \omega_{k+1},$$

using world-model evaluations to update that state. The final planner state determines the returned action sequence. RP1 is the special case in which ω_k is simply the current action plan $\mathbf{a_k}$.

4 Reinforced Planning

Our method follows an actor-critic architecture: a *critic* scores latent states with respect to a goal, and an actor, here a *learned planner*, optimizes the action plan against the critic's final state estimate.

Critic. The critic is a goal-conditioned value function $V_\psi(z_t, z_g)$ that estimates the cost-to-go from latent state z_t to an encoded goal state z_g. Here, lower values correspond to fewer steps to goal and thus signify occupancy of better states. We learn this critic via offline temporal-difference (TD) learning, although it could in theory be any cost-function. During planning, the critic only ever evaluates terminal states $\hat{z}_N$ produced by the rollout operator.

Planner. The planner is a learned operator

$$\mathcal{F}_\theta : \left(\underbrace{\mathbf{a}_k}_{\text{current plan}} \;,\; \underbrace{v_k}_{\text{terminal value}} \;,\; \underbrace{\mathbf{g}_k}_{\text{value gradient}} \right) \longmapsto \underbrace{\mathbf{a}_{k+1}}_{\text{improved plan}} . \tag{3}$$

parametrized by θ, that outputs an improved plan from the current plan, the critic's value at the plan's terminal state, and the plan's value gradient. Starting from an initial plan $\mathbf{a}_0$, planning does three things per step:

$$\textit{roll out:} \quad \hat{z}_N^{(k)} = H_\phi\big(\mathbf{a}_k, z_t\big), \tag{4}$$

$$\textit{evaluate:} \quad v_k = V_\psi\big(\hat{z}_N^{(k)}, z_g\big), \qquad \mathbf{g}_k = \nabla_{\mathbf{a}_k} V_\psi\big(\hat{z}_N^{(k)}, z_g\big), \tag{5}$$

$$\textit{improve:} \quad \mathbf{a}_{k+1} = \mathcal{F}_\theta\big(\mathbf{a}_k,\, v_k,\, \mathbf{g}_k\big). \tag{6}$$

The optimized plan is the final iteration, $\mathbf{a}^\star = \mathbf{a}_K$. While the planner has access to the value and gradient, it is not constrained to follow the plan's gradient $-\mathbf{g}_k$ and can learn when to trust and distrust it. Task-specific information reaches the planner only through v_k and $\mathbf{g}_k$, forcing it to learn a plan-update rule rather than a direct state-and-goal-to-action mapping.

Reinforcing good planning rules. Applying $\mathcal{F}_\theta$ produces an imagined optimization trajectory

$$\mathbf{a}_0 \xrightarrow{\mathcal{F}_\theta} \mathbf{a}_1 \xrightarrow{\mathcal{F}_\theta} \cdots \xrightarrow{\mathcal{F}_\theta} \mathbf{a}_K, \tag{7}$$

where each new plan is obtained by applying the same learned update rule to the preceding plan. The planner is trained to minimize the terminal cost-to-go predicted by the frozen world model H_ϕ and value function V_ψ:

$$\theta^\star = \arg\min_\theta \mathbb{E}_{(z_0,z_g)\sim\mathcal{D}}\left[V_\psi(H_\phi(\mathbf{a}_K, z_t), z_g)\right] + \mathcal{C}, \tag{8}$$

where $\mathcal{C}$ is some regularizer on intermediate plans' value. Update rules that lower the predicted cost-to-go are thereby reinforced in the shared parameters of $\mathcal{F}_\theta$, while updates that raise it are suppressed. The planner therefore learns rules to improve action sequences, not the action sequences themselves.

5 Implementation

As a first realization of a Reinforced Planner, we implement a residual version we call *RP1*. Starting from $\mathbf{a}_0 = \mathbf{0}$, the planner optimizes the action trajectory via

$$\begin{aligned}&\mathcal{F}_\theta\big(\mathbf{a}_k,\, v_k,\, \mathbf{g}_k\big)\\ &\quad= \mathrm{clip}_{[-a_{\max},a_{\max}]}\Big(\underbrace{\mathbf{a}_k}_{\text{prev. plan}} + \underbrace{f_\theta\big(\mathbf{a}_k,\, v_k,\, \mathbf{g}_k\big)}_{\text{residual }\Delta\mathbf{a}_k}\Big),\end{aligned} \tag{9}$$

where $f_\theta\colon \mathcal{A}^N \times \mathbb{R} \times \mathbb{R}^{N\times|a|} \to \mathbb{R}^{N\times|a|}$ is a neural network producing the plan change $\Delta\mathbf{a}_k = f_\theta(\mathbf{a}_k, v_k, \mathbf{g}_k)$.

This residual update $\Delta\mathbf{a}_k$ onto the previous plan $\mathbf{a}_k$ keeps the gradient flow stable to avoid the vanishing gradient problem[14; 13]. The clip is a projection of each plan iterate onto the box $[-a_{\max}, a_{\max}]$ (an ℓ_∞ constraint on the action trajectory, not on the update), bounding actions to $a_{\max}$ standard deviations of the demonstrated distribution so that rollouts stay on the world model's support. We use open-loop planning for our experiments.

We realize the critic as a metric residual network [26] trained with offline value learning through Hindsight Experience Replay[20; 2]. For specifics see Appendix C.

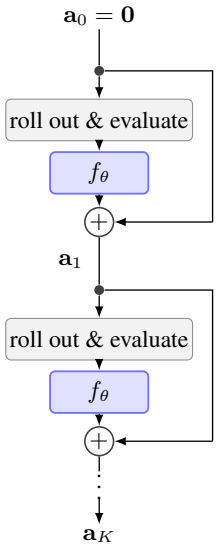


Figure 1: **RP1 visual.**

6 Theoretical Results

We formalize two motivations for Reinforced Planning. First, predictive world-model learning alone does not identify a Euclidean latent geometry that is necessarily suitable for planning. Second, we give an expressivity result for learned planning: under a matched information interface, an expressive learned update rule can perform task-dependent search behavior that no single fixed optimizer configuration can realize across a heterogeneous task distribution. The second result establishes merely a representational advantage of learned search rules rather than a performance guarantee for RP1.

6.1 Latent Norms and Cost-to-go

Recall the encoder E_ϕ and latent transition model h_ϕ from Section 3. For notational simplicity, we write $E_\phi(s)$ for the encoding of the observation generated by state s. Let $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ denote deterministic environment dynamics. We call the latent world model exact when

$$h_\phi(E_\phi(s), a) = E_\phi(\mathcal{T}(s, a)) \qquad \text{for all } (s, a) \in \mathcal{S} \times \mathcal{A}. \tag{10}$$

Theorem 1 (Prediction does not identify Euclidean latent geometry)**.** *Suppose (E_ϕ, h_ϕ) is exact. If the latent displacements from some state s to two goals g_1 and g_2 are linearly independent, then there exist two equally exact latent reparameterizations that reverse which goal is closer to s under Euclidean distance. The ratio between the two distances can be made arbitrarily large.*

Proof sketch. Any invertible linear change of latent coordinates can be absorbed into both the encoder and transition model without changing predictive exactness. By mapping the two goal displacements to separate coordinate axes and stretching either axis, either goal can be made arbitrarily farther than the other. The full proof is given in Appendix B.1. □

Theorem 1 does not imply that latent distance is necessarily a poor planning objective. Rather, it shows that predictive accuracy alone cannot determine whether it is a good one: two equally predictive world models can rank the same candidate goals in opposite orders. Agreement between latent distance and temporal cost-to-go is therefore an additional property that must be learned or imposed separately. We learn this property by training a goal-conditioned critic.

This non-identifiability holds even when the environment is reversible and temporal reachability is symmetric. Appendix B.2 gives the complementary result that temporal reachability can additionally be asymmetric, in which case no symmetric latent norm can represent it exactly.

6.2 Advantage of Learned Planning under Task Heterogeneity

For a fixed world model, action space, planning horizon, and objective, let $x = (z_t, z_g) \sim \mu$ denote a planning task. A planner state $\omega_k \in \Omega_{\mathrm{pl}} \subseteq \mathbb{R}^{d_\omega}$ contains all information carried from one refinement round to the next, including the current candidate plans and any optimizer memory; we assume Ω_{pl} is nonempty and compact. At each round, task-dependent information is exposed through a fixed interface $\mathcal{I}$, shared by the learned and fixed update rules compared below. Starting from a shared initialization ω_0, an update rule F is applied for K rounds:

$$\omega_{F,0}(x) = \omega_0, \qquad \omega_{F,k+1}(x) = F\big(\omega_{F,k}(x), \mathcal{I}\big(x, \omega_{F,k}(x)\big)\big)\,. \tag{11}$$

Its expected loss is

$$\mathcal{L}(F) = \mathbb{E}_{x\sim\mu}\left[\ell\big(x, \omega_{F,K}(x)\big)\right], \tag{12}$$

where ℓ is the cost of the plan returned from the final planner state.

Let $\mathcal{W}$ denote the compact set of feasible planner inputs, and let $\mathfrak{F}_{\mathcal{I}}$ denote the continuous feasible update rules $F : \mathcal{W} \to \Omega_{\mathrm{pl}}$. The precise ambient spaces and regularity conditions are given in Appendix B.3.

Assumption 1 (Universal search-rule approximation)**.** Assume an abstract learned-planner class $\{\mathcal{F}_\theta : \theta \in \Theta\} \subseteq \mathfrak{F}_{\mathcal{I}}$, whose members receive the full feasible input $w \in \mathcal{W}$, can uniformly approximate every rule in $\mathfrak{F}_{\mathcal{I}}$: for every $F \in \mathfrak{F}_{\mathcal{I}}$ and every $\varepsilon > 0$, there exists $\theta \in \Theta$ such that

$$\sup_{w\in\mathcal{W}} \|\mathcal{F}_\theta(w) - F(w)\| < \varepsilon. \tag{13}$$

Now let $\mathcal{X}_1, \ldots, \mathcal{X}_r$ be a measurable partition of the task distribution, with $p_i = \Pr(x \in \mathcal{X}_i) > 0$. For any update rule F, define its regional loss by

$$\mathcal{L}_i(F) = \mathbb{E}\left[\ell\big(x, \omega_{F,K}(x)\big) \,\middle|\, x \in \mathcal{X}_i\right]. \tag{14}$$

Let $\mathcal{B}$ be a family of fixed search configurations. Each $B \in \mathcal{B}$ induces an update rule F_B, and the same configuration is used on every task. We write

$$\mathcal{L}_i(B) = \mathcal{L}_i(F_B), \qquad \mathcal{L}(B) = \sum_{i=1}^{r} p_i \mathcal{L}_i(B). \tag{15}$$

Theorem 2 (Strict advantage under task heterogeneity)**.** *Under Assumption 1 and the regularity, regional incompatibility, and interface-composability conditions stated in Appendix B.3,*

$$\inf_{\theta\in\Theta} \mathcal{L}(\mathcal{F}_\theta) \le \sum_{i=1}^{r} p_i \min_{B\in\mathcal{B}} \mathcal{L}_i(B) < \min_{B\in\mathcal{B}} \mathcal{L}(B). \tag{16}$$

Thus, a sufficiently expressive learned planner can strictly outperform every single fixed search configuration by adapting its update behavior across task regions through the shared interface.

Note that theorem 2 is an expressivity result, not a claim that learning an optimizer necessarily yields better empirical performance. Applying the result specifically to RP1 would require additional regularity conditions ensuring that $\mathbf{g} = \nabla_{\mathbf{a}} V_\psi(H_\phi(\mathbf{a}, z_t), z_g)$ is well-defined and continuous on the relevant domain.

7 EXPERIMENTS

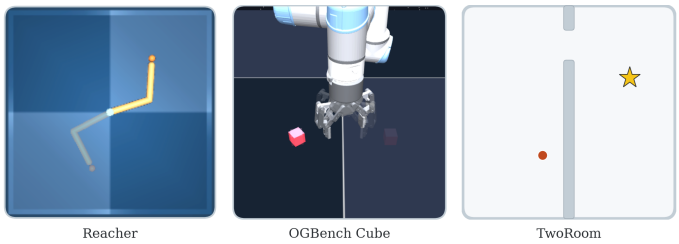


Figure 2: **Experiment environments.** In Reacher (left) the agent moves a two-link arm to a goal configuration, here shaded. In OGBench Cube (middle) a robot arm picks up a cube and moves it to a goal position, also shaded. In TwoRoom (right) an agent navigates to a goal position, (here the star).

Evaluation design. We evaluate RP1 in three visual-control domains, TwoRoom, Reacher, and OGBench Cube, on two world-model backbones: LeWorldModel (LeWM) and PLDM. Each backbone is trained separately for each environment, so every (environment, backbone) pair has its own latent space and dynamics. RP1 is trained per pair for the same reason: its inputs, the value v_k and gradient $\mathbf{g}_k$, are computed through a specific world model and critic, and their scale and structure change with the latent space. Thus, moving to a new world model base requires retraining the planner's weights but no new configuration search. We use benchmarks from the StableWorldModel environment [28].

Baselines and controlled comparisons. We compare RP1 against three popular hand-designed planning algorithms and two partly-learned hybrid planners. Essentially all recent latent-planning systems report using one of the hand-designed planners or a close variant [7; 11; 53; 42; 37]. We evaluate each with both latent distance $\|\hat{z}_N - z_g\|_2^2$ and the learned objective $V(\hat{z}_N, z_g)$. The hybrid planners DMPO and L2O-MPC use a learned critic, following their original design [39; 38].

Evaluation metrics. We report task success and planning compute cost, measured as the number of world-model rollouts per decision. All planners use identical action chunking: each planned action comprises five primitive actions, and each planner optimizes a sequence of five such chunks, corresponding to a horizon of 25 primitive actions. All methods therefore plan over the same horizon in the same normalized action space. Full evaluation details are provided in Appendix D.

7.1 TWOROOM

TwoRoom tests whether model-based agents are capable of appropriate planning when geometric proximity differs from temporal reachability. The agent must pass through a doorway to reach the opposite room, so states that are geometrically close across the wall might still require a long detour.

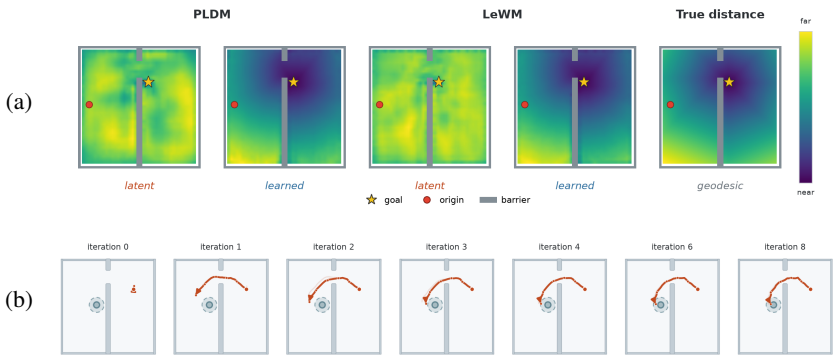


Figure 3: **RP1 in TwoRoom.** (a) The learned critic better captures temporal cost-to-go than latent L_2 distance. (b) RP1 iteratively refines its plan, with later updates focusing on fine corrections to the final actions. Additional visualizations are provided in Appendix D.2.

We find that the latent-distance objective does not capture distance-to-goal in the queried world-models, whereas a learned value function saturates the benchmark across implemented planners at 25−step horizon. Once planners are given the learned value critic, their performance largely converges: most methods reach near-saturated success, despite using very different search rules and compute budgets.

TwoRoom

		LeWM		PLDM	
planner	rollouts	25 steps	100 steps	25 steps	100 steps
latent L_2 *(value critic)*					
CEM	9000	84.0 (**100.0**)	13.3 (73.0)	93.3 (**100.0**)	52.0 (49.3)
MPPI	9000	70.7 (95.3)	20.0 (68.0)	64.0 (95.7)	33.3 (53.3)
Adam	3000	94.7 (96.3)	24.0 (68.0)	90.7 (94.0)	42.0 (53.0)
DMPO	256	73.3	69.3	76.7	44.0
L2O-MPC	256	98.0	92.7	99.3	46.7
RP1 (ours)	9	**100.0**	**99.3**	98.0	**94.0**

Table 1: **Success rate (%).** Main number: latent L_2; in parentheses: the learned value critic RP1 deploys. **Bold** marks the strongest baseline per column.

7.2 REACHER

Reacher is a two-link arm under torque control, observed only as visual frames. The task is to bring both joints into a target configuration. Success follows the benchmark's first-hit convention at a loose and a tight tolerance (τ=0.1 and τ=0.05 rad). As in all domains, planners optimize five blocks of five primitive actions, so the planning horizon exactly covers the nominal 25-step distance to the goal.

Reacher

		LeWM		PLDM	
planner	rollouts	τ=.1	τ=.05	τ=.1	τ=.05
latent L_2 *(value critic)*					
CEM	9000	98.7 (88.0)	80.3 (63.3)	96.7 (75.3)	80.0 (56.3)
MPPI	9000	90.0 (89.0)	69.3 (70.3)	89.3 (86.0)	63.3 (65.3)
Adam	3000	94.0 (92.7)	66.0 (71.7)	94.3 (92.7)	66.0 (68.7)
DMPO	256	40.7	20.7	36.0	19.3
L2O-MPC	256	68.0	46.7	67.3	39.3
RP1 (ours)	9	**100.0**	**95.3**	**99.3**	**87.3**

Table 2: **First-hit success (%).** Goal tolerance τ (rad). Main number: latent L_2; in parentheses: the learned value critic RP1 deploys. **Bold** marks the strongest baseline per column.

Even this near-saturated task discriminates between planners once the tolerance is tightened. At τ=0.1, the strongest planners all bring the arm into the neighborhood of the goal, with success rates differing by only a few percentage points. RP1 does so using three orders of magnitude fewer world-model rollouts. All methods degrade with decreased goal tolerance τ, but RP1 retains the highest success rate in every column; relative to the strongest baseline, its margin increases from approximately one percentage point at τ=0.1 to 15 points on LeWM and 7 points on PLDM at τ=0.05.

7.3 OGBENCH CUBE

OGBench Cube [34] is a vision-based manipulation benchmark: a robot arm must pick up a cube and place it at a goal position, observed only from pixels, with goals placed 25 or 100 primitive

steps away ($h25$, $h100$). Alongside the regular OGBench score we report a *hard* score, the same runs normalized against the measured no-op floor f as $(s - f)/(100 - f) \cdot 100$, which measures the fraction of headroom above doing nothing that a planner actually converts. The hard score is our primary number; easy is kept for comparability with the benchmark's convention.

OGBench Cube.

	LeWM				PLDM			
	25 steps		100 steps		25 steps		100 steps	
planner (roll.)	easy	hard	easy	hard	easy	hard	easy	hard
latent L_2 *(value-critic)*								
CEM 9000	74.0(80.0)	40.9(54.5)	58.0(77.7)	23.2(59.2)	62.7(75.3)	15.2(43.9)	58.7(69.3)	24.5(43.8)
MPPI 9000	56.7(67.0)	1.6(25.0)	46.0(58.3)	1.2(23.7)	60.0(65.3)	9.1(21.1)	47.3(53.0)	3.6(14.0)
Adam 3000	74.0(77.3)	40.9(48.4)	57.3(70.7)	21.9(46.4)	63.3(66.0)	16.6(22.7)	55.3(57.0)	18.2(21.3)
DMPO 256	66.0	22.7	54.0	15.9	61.3	12.0	50.0	8.5
L2O-MPC 256	64.0	18.2	54.0	15.9	56.0	0.0	45.3	0.0
RP1 (ours) 9	**92.7**	**83.4**	**85.3**	**73.1**	**85.3**	**66.6**	**83.7**	**70.2**

Table 3: **Success rate (%).** Easy is the raw rate; hard is normalized against the no-op floor (56.0 at 25 steps, 45.3 at 100), clipped at 0 for methods below it. **Bold** marks the best entry per column.

7.4 World-Model Hallucination and Dyna Finetuning

Training the planner through a frozen world model lets it exploit model error. Inspecting RP1's failure episodes in OGBench Cube, we found the world model *hallucinating* contact outcomes: for LeWM, grasps that miss the cube are nevertheless predicted "magically" to attach it to the arm. No improvement in search can fix such hallucinations. We therefore correct the model rather than the planner: one Dyna iteration [45] deploys the trained planner, collects its (failure) rollouts, finetunes the world model on them, and retrains the planner (Appendix C.3).

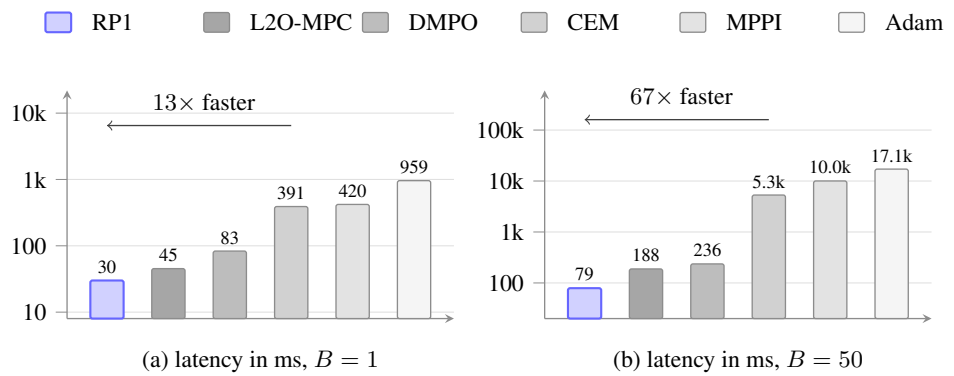
	LeWM		PLDM	
	easy	hard	easy	hard
RP1 (pretrained)	92.7	83.4	85.3	66.6
RP1_{Dyna} (finetuned)	**95.0**	**88.6**	**93.3**	**84.8**

Table 4: **Effect of Dyna at $h25$ (%).** Rollouts are collected at both goal horizons by the deployed planners and the locked configuration is retrained on the finetuned model; we run two rounds and report the one chosen on validation (round 1 for LeWM, round 2 for PLDM)..

One iteration recovers a large part of the exploitation gap, and we found empirically that the "grasp-and-miss" hallucinations were significantly reduced in LeWM. However, despite mitigating exploitation, characterizing when it recurs remains open.

7.5 Planning Speed

We measure end-to-end planning latency on OGBench Cube at the 25-step goal offset with LeWM, including the complete computation from the input latents to the returned action plan, including RP1's backward passes through the world model and critic. All methods run in fp32 on a single NVIDIA H200 and are benchmarked using both CUDA-graph-captured and eager execution, with the faster mean reported. We consider one planner running alone ($B = 1$) and 50 independent planners running concurrently on the same GPU ($B = 50$), representing multiple control loops sharing one accelerator. Hand-designed planners use the learned value function as their objective.



(a) latency in ms, $B = 1$ (b) latency in ms, $B = 50$

Figure 4: **End-to-end planning latency.** On OGBench Cube with LeWM (one NVIDIA H200, fp32), with RP1 $13\times$ faster than CEM for one planner and $67\times$ faster for 50 concurrent planners.

The $1{,}000\times$ reduction in world-model rollouts does not translate one-for-one into single-planner latency because the GPU can evaluate many of a sampling planner's candidate trajectories in parallel. Nevertheless, RP1 completes a single planning request $13\times$ faster than the critic-based CEM alternative, the strongest conventional planner by success rate. The advantage grows substantially under concurrent inference: RP1 processes 50 concurrently deployed planners $67\times$ faster than CEM. RP1 also remains $3\times$ faster than DMPO and $2.4\times$ faster than L2O-MPC in this setting. Thus, the rollout reduction becomes most consequential when one accelerator serves several control loops, such as multiple robot arms planning in tandem.

8 Discussion

Our results support both motivating hypotheses: where geometric proximity differs from temporal reachability, a learned critic resolves failures that more search does not (Sec. 7.1); and where the objective is already informative, a learned update rule matches or exceeds hand-designed planners with two to three orders of magnitude fewer world-model queries.

Limitations. Because RP1 is trained through a frozen world model, it can exploit model errors; Dyna finetuning (Sec. 7.4) reduces but does not eliminate this. Varying the number of refinement steps K gave improvements up to $K = 8$, but for higher K did not noticeably change RP1's success, and for the baselines we use the budgets of prior work [53; 27; 39; 38] rather than tracing their success as a function of budget. Unlike hand-designed planners, RP1 must be retrained for each world model and goal horizon. We do not compare against amortized policies trained through the same critic, as we only aim to compare inference-time model-based planners. Checkpoint selection uses simulator evaluation on held-out tasks, a common practice in offline RL [32; 21]; Appendix D.1 reports its cost.

9 AI Use Statement

Generative Artificial Intelligence aided in the development of this paper. Specifically, it helped with the polishing of prose and mathematics, literature search and code-writing. We have reviewed and verified any LLM-generated code. All ideas behind the method are products of the author's human minds.

10 Acknowledgements

References

[1] Marcin Andrychowicz, Misha Denil, Sergio Gómez, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[3] Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15619–15629, 2023.

[4] Thomas Bush, Stephen Chung, Usman Anwar, Adrià Garriga-Alonso, and David Krueger. Interpreting emergent planning in model-free reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pages 82115–82197, 2025.

[5] Stephen Chung, Ivan Anokhin, and David Krueger. Thinker: Learning to plan and act. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2307.14993.

[6] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. *Advances in Neural Information Processing Systems*, 31, 2018. Extended version "World Models", arXiv:1803.10122.

[7] 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 (ICML)*, pages 2555–2565, 2019.

[8] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination, 2020. URL `https://arxiv.org/abs/1912.01603`.

[9] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640:647–653, 2025.

[10] Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. In *International Conference on Machine Learning (ICML)*, 2022.

[11] Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations (ICLR)*, 2024.

[12] Kristian Hartikainen, Xinyang Geng, Tuomas Haarnoja, and Sergey Levine. Dynamical distance learning for semi-supervised and unsupervised skill discovery. In *International Conference on Learning Representations (ICLR)*, 2020.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[14] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. Master's thesis, Technische Universität München, 1991.

[15] Peter J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964.

[16] Taher Jafferjee, Ehsan Imani, Erik Talvitie, Martha White, and Michael Bowling. Hallucinating value: A pitfall of dyna-style planning with imperfect environment models. *arXiv preprint arXiv:2006.04363*, 2020.

[17] Michael Janner, Yilun Du, Joshua B. Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis, 2022. URL `https://arxiv.org/abs/2205.09991`.

[18] Adam Johnson and A. David Redish. Neural ensembles in CA3 transiently encode paths forward of the animal at a decision point. *Journal of Neuroscience*, 27(45):12176–12189, 2007. doi: 10.1523/JNEUROSCI.3761-07.2007.

[19] Leslie Pack Kaelbling. Learning to achieve goals. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1993.

[20] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations (ICLR)*, 2022.

[21] Vladislav Kurenkov and Sergey Kolesnikov. Showing your offline reinforcement learning work: Online evaluation budget matters. In *International Conference on Machine Learning*, 2022.

[22] Yann LeCun. A path towards autonomous machine intelligence. `https://openreview.net/forum?id=BZ5a1r-kVsf`, 2022. Version 0.9.2.

[23] Ke Li and Jitendra Malik. Learning to optimize. In *International Conference on Learning Representations*, 2017.

[24] Liangyu Li, Shengzhi Wang, and Qingwen Liu. Beyond euclidean proximity: Repairing latent world models with horizon-matched trajectory reachability metrics, 2026. URL `https://arxiv.org/abs/2605.22164`.

[25] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

[26] Bo Liu, Yihao Feng, Qiang Liu, and Peter Stone. Metric residual networks for sample efficient goal-conditioned reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

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

[28] Lucas Maes, Quentin Le Lidec, Luiz Facury, Nassim Massaudi, Ayush Chaurasia, Francesco Capuano, Richard Gao, Taj Gillin, Dan Haramati, Damien Scieur, Yann LeCun, and Randall Balestriero. stable-worldmodel: A platform for reproducible world modeling research and evaluation, 2026. URL `https://arxiv.org/abs/2605.21800`.

[29] Joseph Marino, Alexandre Piché, Alessandro Davide Ialongo, and Yisong Yue. Iterative amortized policy optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[31] Whitney K. Newey and James L. Powell. Asymmetric least squares estimation and testing. *Econometrica*, 55(4):819–847, 1987.

[32] Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyu Wang, and Nando de Freitas. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*, 2020.

[33] Seohong Park, Dibya Ghosh, Benjamin Eysenbach, and Sergey Levine. Hiql: Offline goal-conditioned rl with latent states as actions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[34] Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. OGBench: Benchmarking offline goal-conditioned RL. In *International Conference on Learning Representations (ICLR)*, 2025.

[35] Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sébastien Racanière, David Reichert, Théophane Weber, Daan Wierstra, and Peter Battaglia. Learning model-based planning from scratch. *arXiv preprint arXiv:1707.06170*, 2017.

[36] Brad E. Pfeiffer and David J. Foster. Hippocampal place-cell sequences depict future paths to remembered goals. *Nature*, 497(7447):74–79, 2013. doi: 10.1038/nature12112.

[37] Jyothir S V, Siddhartha Jalagam, Yann LeCun, and Vlad Sobal. Gradient-based planning with world models. *arXiv preprint arXiv:2312.17227*, 2023.

[38] Jacob Sacks and Byron Boots. Learning to optimize in model predictive control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 10549–10556, 2022.

[39] Jacob Sacks, Rwik Rana, Kevin Huang, Alex Spitzer, Guanya Shi, and Byron Boots. Deep model predictive optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[40] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning (ICML)*, 2015.

[41] James E. Smith and Robert L. Winkler. The optimizer's curse: Skepticism and postdecision surprise in decision analysis. *Management Science*, 52(3):311–322, 2006. doi: 10.1287/mnsc.1050.0451.

[42] Vlad Sobal, Wancong Zhang, Kyunghyun Cho, Randall Balestriero, Tim G. J. Rudner, and Yann LeCun. Learning from reward-free offline data: A case for planning with latent dynamics models. *arXiv preprint arXiv:2502.14819*, 2025.

[43] Armin Sommer. Planning as emergent behavior in reinforcement learning with relational hidden states, 2026. URL `https://arxiv.org/abs/2607.18589`.

[44] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *International Conference on Machine Learning (ICML)*, pages 4732–4741, 2018.

[45] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991. doi: 10.1145/122344.122377.

[46] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

[47] Erik Talvitie. Self-correcting models for model-based reinforcement learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI)*, pages 2597–2603, 2017.

[48] Mohammad Taufeeque, Aaron Tucker, Adam Gleave, and Adrià Garriga-Alonso. Path channels and plan extension kernels: A mechanistic description of planning in a sokoban rnn. In *International Conference on Learning Representations*, volume 2026, pages 134244–134277, 2026.

[49] Edward C. Tolman. Cognitive maps in rats and men. *Psychological Review*, 55(4):189–208, 1948.

[50] Tongzhou Wang and Phillip Isola. Improved representation of asymmetrical distances with interval quasimetric embeddings. In *NeurIPS Workshop on Symmetry and Geometry in Neural Representations*, 2022.

[51] Tongzhou Wang, Antonio Torralba, Phillip Isola, and Amy Zhang. Optimal goal-reaching reinforcement learning via quasimetric learning. In *International Conference on Machine Learning (ICML)*, 2023.

[52] Tongzhou Wang, Antonio Torralba, Phillip Isola, and Amy Zhang. Optimal goal-reaching reinforcement learning via quasimetric learning. In *International Conference on Machine Learning (ICML)*, pages 36411–36430, 2023.

[53] Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning. In *International Conference on Machine Learning (ICML)*, 2025. arXiv:2411.04983.

A CODEBASE

The code is available at `https://anonymous.4open.science/r/RLP_original-0A0D`.

B PROOFS OF THEORETICAL RESULTS

We formalize two motivations for Reinforced Planning. First, predictive world-model learning does not determine a Euclidean latent geometry suitable for planning. Second, an abstract learned neural planner can adapt its optimization rule to the task, whereas conventional optimizers use one configuration across the task distribution.

B.1 PROOF OF THEOREM 1

Proof. Let $A \in \mathbb{R}^{d\times d}$ be invertible and define

$$E_A(s) = AE_\phi(s), \qquad h_A(z,a) = Ah_\phi(A^{-1}z, a). \tag{17}$$

Then

$$\begin{aligned} h_A(E_A(s),a) &= Ah_\phi(A^{-1}AE_\phi(s),a) && (18)\\ &= Ah_\phi(E_\phi(s),a) && (19)\\ &= AE_\phi(\mathcal{T}(s,a)) && (20)\\ &= E_A(\mathcal{T}(s,a)). && (21) \end{aligned}$$

Thus, (E_A, h_A) is exact whenever (E_ϕ, h_ϕ) is exact. The same argument applied recursively shows that all multi-step trajectories remain exactly predicted under the transformed coordinates.

Now define

$$u = E_\phi(g_1) - E_\phi(s), \qquad v = E_\phi(g_2) - E_\phi(s). \tag{22}$$

Because u and v are linearly independent, there exists an invertible matrix B such that

$$Bu = e_1, \qquad Bv = e_2, \tag{23}$$

where e_1 and e_2 are the first two standard basis vectors.

For any $R > 1$, define

$$\begin{aligned} A_1 &= \mathrm{diag}(R, 1, \ldots, 1)B, && (24)\\ A_2 &= \mathrm{diag}(1, R, 1, \ldots, 1)B. && (25) \end{aligned}$$

Both matrices are invertible and therefore induce exact latent world models. Under the first transformation,

$$\frac{\|E_{A_1}(g_1) - E_{A_1}(s)\|_2}{\|E_{A_1}(g_2) - E_{A_1}(s)\|_2} = R, \tag{26}$$

whereas under the second,

$$\frac{\|E_{A_2}(g_1) - E_{A_2}(s)\|_2}{\|E_{A_2}(g_2) - E_{A_2}(s)\|_2} = \frac{1}{R}. \tag{27}$$

The two exact world models therefore induce opposite Euclidean distance orderings. Since R is arbitrary, the separation between the distances can be made arbitrarily large. □

B.2 TEMPORAL REACHABILITY AS A DIRECTED DISTANCE

Consider a deterministic controlled system with state space $\mathcal{S}$. Define

$$d^\star(s,g) := \inf\left\{T \in \mathbb{N}_0 : \text{some length-}T \text{ action sequence takes } s \text{ to } g\right\}, \tag{28}$$

with $d^\star(s,g) = +\infty$ when g is unreachable from s.

Proposition 1 (Temporal reachability is an extended directed quasimetric)**.** *For all states s, y, g,*

$$d^\star(s,g) \geq 0, \qquad d^\star(s,g) = 0 \iff s = g, \qquad d^\star(s,g) \leq d^\star(s,y) + d^\star(y,g). \tag{29}$$

However, $d^\star(s,g)$ need not equal $d^\star(g,s)$. Consequently, when temporal reachability is asymmetric, no symmetric distance such as a latent norm $\|E(s) - E(g)\|_2$ can represent it exactly on all ordered state pairs.

Proof. The empty action sequence takes each state to itself, so $d^\star(s,s) = 0$. Conversely, a length-zero sequence cannot change the state, so $d^\star(s,g) = 0$ implies $s = g$. Nonnegativity follows because action sequence lengths belong to $\mathbb{N}_0$.

The triangle inequality is immediate if either $d^\star(s,y)$ or $d^\star(y,g)$ is infinite. Otherwise, concatenate a shortest sequence from s to y with a shortest sequence from y to g. The resulting sequence takes s to g and has length $d^\star(s,y) + d^\star(y,g)$.

Finally, consider two states for which an action takes s to g, but no action sequence returns from g to s. Then $d^\star(s,g) = 1$ while $d^\star(g,s) = +\infty$. Because every symmetric distance assigns the same value to (s,g) and (g,s), it cannot represent $d^\star$ exactly in this case. □

B.3 Formal Statement and Proof of Theorem 2

We first make the ambient spaces and regularity conditions explicit. Let

$$\mathcal{Z} \subseteq \mathbb{R}^{d_z}, \qquad \mathcal{X} \subseteq \mathcal{Z} \times \mathcal{Z} \subseteq \mathbb{R}^{2d_z}, \qquad \Omega_{\mathrm{pl}} \subseteq \mathbb{R}^{d_\omega}, \qquad \mathcal{Y} \subseteq \mathbb{R}^{d_y}. \tag{30}$$

Here, $\mathcal{X}$ is the task space, with $x = (z_t, z_g) \in \mathcal{X}$, Ω_{pl} is the planner-state space, whose coordinates may include the current plan and bounded optimizer memory, and $\mathcal{Y}$ is the output space of the planner interface. We assume that $\mathcal{X}$ and Ω_{pl} are nonempty and compact, and equip all finite-dimensional spaces and product spaces with their Euclidean norms.

Let μ be a probability distribution supported on $\mathcal{X}$, let $\omega_0 \in \Omega_{\mathrm{pl}}$ be the common planner initialization, and let $K < \infty$ be the number of refinement rounds. Assume that the abstract matched interface $\mathcal{I} : \mathcal{X} \times \Omega_{\mathrm{pl}} \to \mathcal{Y}$ and $\ell : \mathcal{X} \times \Omega_{\mathrm{pl}} \to \mathbb{R}$ are continuous.

Define the set of feasible planner inputs by

$$\mathcal{W} = \{(\omega, \mathcal{I}(x,\omega)) : x \in \mathcal{X},\ \omega \in \Omega_{\mathrm{pl}}\} \subseteq \mathbb{R}^{d_\omega + d_y}. \tag{31}$$

Because $\mathcal{X} \times \Omega_{\mathrm{pl}}$ is compact and $(x,\omega) \mapsto (\omega, \mathcal{I}(x,\omega))$ is continuous, $\mathcal{W}$ is compact.

Let

$$\mathfrak{F}_{\mathcal{I}} = \{F : \mathcal{W} \to \Omega_{\mathrm{pl}} \ :\ F \text{ is continuous}\} \tag{32}$$

be the class of continuous feasible search rules. For any $F \in \mathfrak{F}_{\mathcal{I}}$, set $\omega_{F,0}(x) = \omega_0$, and, for $k = 0, \ldots, K-1$,

$$\omega_{F,k+1}(x) = F\big(\omega_{F,k}(x), \mathcal{I}\big(x, \omega_{F,k}(x)\big)\big). \tag{33}$$

Since every rule maps $\mathcal{W}$ into Ω_{pl}, all planner states remain feasible.

The expected loss of F is

$$\mathcal{L}(F) = \mathbb{E}_{x\sim\mu}\left[\ell\big(x, \omega_{F,K}(x)\big)\right]. \tag{34}$$

Assumption 1 states that the abstract learned-planner class $\{\mathcal{F}_\theta : \theta \in \Theta\} \subseteq \mathfrak{F}_{\mathcal{I}}$ receives the full feasible input $w \in \mathcal{W}$ and that, for every $F \in \mathfrak{F}_{\mathcal{I}}$ and every $\varepsilon > 0$, there exists $\theta \in \Theta$ satisfying

$$\sup_{w\in\mathcal{W}} \|\mathcal{F}_\theta(w) - F(w)\| < \varepsilon. \tag{35}$$

Let $\mathcal{X}_1, \ldots, \mathcal{X}_r$ be a measurable partition of $\mathcal{X}$, up to sets of μ-measure zero, with

$$p_i = \mu(\mathcal{X}_i) > 0. \tag{36}$$

For every $F \in \mathfrak{F}_{\mathcal{I}}$, define

$$\mathcal{L}_i(F) = \mathbb{E}\left[\ell\big(x, \omega_{F,K}(x)\big) \,\middle|\, x \in \mathcal{X}_i\right]. \tag{37}$$

Let

$$\mathcal{B} \subseteq \mathbb{R}^{d_B} \tag{38}$$

be a nonempty compact family of fixed search configurations. Each $B \in \mathcal{B}$ induces a rule $F_B \in \mathfrak{F}_{\mathcal{I}}$, with the same configuration B used on every task. Define

$$\mathcal{L}_i(B) = \mathcal{L}_i(F_B), \qquad \mathcal{L}(B) = \sum_{i=1}^{r} p_i \mathcal{L}_i(B), \tag{39}$$

and assume that $B \mapsto \mathcal{L}_i(B)$ is continuous for every region i. Consequently, the regional minimum

$$b_i := \min_{B \in \mathcal{B}} \mathcal{L}_i(B) \tag{40}$$

exists for every i.

Assumption 2 (Regional incompatibility)**.** No fixed configuration minimizes every regional loss:

$$\bigcap_{i=1}^{r} \operatorname*{argmin}_{B \in \mathcal{B}} \mathcal{L}_i(B) = \varnothing. \tag{41}$$

Assumption 3 (Interface composability)**.** There exists a continuous feasible rule $F^\star \in \mathfrak{F}_{\mathcal{I}}$ that attains the best fixed-configuration loss in every region:

$$\mathcal{L}_i(F^\star) = b_i = \min_{B \in \mathcal{B}} \mathcal{L}_i(B), \qquad i = 1, \dots, r. \tag{42}$$

Theorem 3 (Strict advantage under task heterogeneity; restatement of Theorem 2)**.** *Under Assumption 1, Assumption 2, and Assumption 3,*

$$\inf_{\theta \in \Theta} \mathcal{L}(\mathcal{F}_\theta) \le \sum_{i=1}^{r} p_i \min_{B \in \mathcal{B}} \mathcal{L}_i(B) < \min_{B \in \mathcal{B}} \mathcal{L}(B). \tag{43}$$

Proof. By Assumption 3, there exists $F^\star \in \mathfrak{F}_{\mathcal{I}}$ such that $\mathcal{L}_i(F^\star) = b_i$ for every i. Since the regions partition the task distribution,

$$\mathcal{L}(F^\star) = \sum_{i=1}^{r} p_i \mathcal{L}_i(F^\star) = \sum_{i=1}^{r} p_i b_i. \tag{44}$$

We next show that the neural-planner class can approach this loss. For each integer $m \ge 1$, apply Assumption 1 with $\varepsilon = 1/m$. This gives a parameter $\theta_m \in \Theta$ satisfying

$$\sup_{w \in \mathcal{W}} \|\mathcal{F}_{\theta_m}(w) - F^\star(w)\| < \frac{1}{m}. \tag{45}$$

For brevity, write

$$\omega_{m,k}(x) = \omega_{\mathcal{F}_{\theta_m},k}(x), \qquad \omega_k^\star(x) = \omega_{F^\star,k}(x). \tag{46}$$

We prove by induction that, for every fixed $k \le K$,

$$\sup_{x \in \mathcal{X}} \|\omega_{m,k}(x) - \omega_k^\star(x)\| \longrightarrow 0 \qquad \text{as } m \to \infty. \tag{47}$$

The claim holds for $k = 0$, because all planners share the initialization ω_0. Suppose that it holds at round k. Define

$$w_{m,k}(x) = \big(\omega_{m,k}(x), \mathcal{I}\big(x, \omega_{m,k}(x)\big)\big), \tag{48}$$

$$w_k^\star(x) = \big(\omega_k^\star(x), \mathcal{I}\big(x, \omega_k^\star(x)\big)\big). \tag{49}$$

Continuity of $\mathcal{I}$ on the compact set $\mathcal{X} \times \Omega_{\mathrm{pl}}$ implies uniform continuity. Therefore, the induction hypothesis gives

$$\sup_{x \in \mathcal{X}} \|w_{m,k}(x) - w_k^\star(x)\| \longrightarrow 0. \tag{50}$$

Using the planner recursion and adding and subtracting $F^\star(w_{m,k}(x))$, we obtain

$$\sup_{x \in \mathcal{X}} \big\|\omega_{m,k+1}(x) - \omega_{k+1}^\star(x)\big\| \tag{51}$$

$$\le \sup_{x \in \mathcal{X}} \big\|\mathcal{F}_{\theta_m}\big(w_{m,k}(x)\big) - F^\star\big(w_{m,k}(x)\big)\big\| + \sup_{x \in \mathcal{X}} \big\|F^\star\big(w_{m,k}(x)\big) - F^\star\big(w_k^\star(x)\big)\big\|. \tag{52}$$

The first term is at most $1/m$ by Eq. 45. The second converges to zero because $F^\star$ is uniformly continuous on the compact set $\mathcal{W}$ and Eq. 50 holds. This proves Eq. 47 for every finite $k \le K$.

Because ℓ is continuous on the compact set $\mathcal{X} \times \Omega_{\mathrm{pl}}$, it is uniformly continuous. Applying Eq. 47 at $k = K$ therefore yields

$$\sup_{x \in \mathcal{X}} \left| \ell\big(x, \omega_{m,K}(x)\big) - \ell\big(x, \omega_K^\star(x)\big) \right| \longrightarrow 0. \tag{53}$$

Consequently,

$$\mathcal{L}(\mathcal{F}_{\theta_m}) \longrightarrow \mathcal{L}(F^\star). \tag{54}$$

Together with Eq. 44, this gives

$$\inf_{\theta \in \Theta} \mathcal{L}(\mathcal{F}_\theta) \le \sum_{i=1}^{r} p_i b_i. \tag{55}$$

It remains to show that every single fixed configuration has strictly larger expected loss. Define its excess over the regional minima by

$$\Delta(B) = \sum_{i=1}^{r} p_i \big(\mathcal{L}_i(B) - b_i\big). \tag{56}$$

Every term in this sum is nonnegative. By Assumption 2, each $B \in \mathcal{B}$ is strictly suboptimal in at least one region. Since every $p_i > 0$,

$$\Delta(B) > 0 \qquad \text{for every } B \in \mathcal{B}. \tag{57}$$

The function Δ is continuous because it is a finite weighted sum of the continuous functions $\mathcal{L}_i$. Since $\mathcal{B}$ is compact, Δ attains its minimum. Its pointwise strict positivity implies

$$\eta := \min_{B \in \mathcal{B}} \Delta(B) > 0. \tag{58}$$

Hence

$$\min_{B \in \mathcal{B}} \mathcal{L}(B) = \min_{B \in \mathcal{B}} \left[\sum_{i=1}^{r} p_i b_i + \Delta(B) \right] \tag{59}$$

$$= \sum_{i=1}^{r} p_i b_i + \eta \tag{60}$$

$$> \sum_{i=1}^{r} p_i b_i. \tag{61}$$

Combining Eq. 55 with Eq. 61 proves

$$\inf_{\theta \in \Theta} \mathcal{L}(\mathcal{F}_\theta) \le \sum_{i=1}^{r} p_i \min_{B \in \mathcal{B}} \mathcal{L}_i(B) < \min_{B \in \mathcal{B}} \mathcal{L}(B). \tag{62}$$

□

C METHOD DETAILS

C.1 VALUE LEARNING

For each environment and world model, we train a separate goal-conditioned cost-to-go function [19; 40]

$$V_\psi(z, z_g)\colon \mathcal{Z} \times \mathcal{Z} \to \mathbb{R}_{\geq 0}. \tag{63}$$

Lower values represent shorter predicted temporal distance [12] to the goal, as we assume a cost of 1 per step. The world-model encoder is frozen, and the value function is trained entirely from cached offline latents. The value is represented by a metric residual network [26; 50],

$$V_\psi(z, z_g) = \|u_\psi(z) - u_\psi(z_g)\|_2 + \max_j \mathrm{ReLU}\left(v_{\psi,j}(z_g) - v_{\psi,j}(z)\right). \tag{64}$$

Here u_ψ is the first half of the latent vector the critic head computes and v_ψ is the second half. The first term is symmetric, while the second permits directed temporal distance [51]. For each update, we sample an anchor z_t, an n-step successor $z_{t+n_{\mathrm{eff}}}$, and a hindsight goal [2] z_g, where $n_{\mathrm{eff}} = \min\{n, T_{\mathrm{episode}} - t\}$. In-episode goals are sampled from future states with temporal offsets balanced across the available episode horizon. Cross-episode goals are additionally sampled to train long-range state pairs. If an in-episode goal lies within the backup window, its exact temporal distance δ is used. Otherwise, the target is bootstrapped with an n-step backup [46]:

$$y_t = \begin{cases} \delta, & \delta \leq n_{\mathrm{eff}}, \\ c_\gamma(n_{\mathrm{eff}}) + \gamma^{n_{\mathrm{eff}}} \bar{V}_{\bar{\psi}}(z_{t+n_{\mathrm{eff}}}, z_g), & \text{otherwise}, \end{cases} \tag{65}$$

with

$$c_\gamma(n) = \sum_{i=0}^{n-1} \gamma^i = \begin{cases} n, & \gamma = 1, \\ \dfrac{1-\gamma^n}{1-\gamma}, & \gamma < 1. \end{cases} \tag{66}$$

The target parameters [30] are updated by Polyak averaging [25],

$$\bar{\psi} \leftarrow (1-\eta)\bar{\psi} + \eta\psi. \tag{67}$$

Following implicit Q-learning [20; 33], the value function is trained by asymmetric expectile regression [31], replacing the squared penalty with a Huber penalty [15] for robustness:

$$\mathcal{L}_V(\psi) = \mathbb{E}_{(z_t, z_g)\sim\mathcal{D}}\Big[\big|\tau - \mathbf{1}[V_\psi(z_t, z_g) - y_t > 0]\big|\, \ell_{\mathrm{Huber}}(V_\psi(z_t, z_g) - y_t)\Big]. \tag{68}$$

Since V_ψ is a cost-to-go rather than a return, we use $\tau < 0.5$: the weight on overestimation exceeds the weight on underestimation, so V_ψ regresses toward a lower expectile of the target distribution, approximating the shortest temporal distance realizable in the data rather than the behavior-policy average.

C.2 RP1 TRAINING

RP1 is trained entirely offline while the world-model encoder and dynamics predictor remain frozen. For each world model, the planner is trained from a stride-five latent cache aligned with five-step action blocks. RP1 consists of three fully-connected layers with ReLU activations and hidden width 512, mapping $\mathbb{R}^{2N|a|+1} \to \mathbb{R}^{512} \to \mathbb{R}^{512} \to \mathbb{R}^{N|a|}$, where the input concatenates the flattened plan $\mathbf{a}_k \in \mathbb{R}^{N|a|}$, its value gradient $\mathbf{g}_k \in \mathbb{R}^{N|a|}$, and the scalar terminal value v_k, and the output is the residual plan update. For example in OGBench Cube, with a planning horizon of $N = 5$ action blocks and $|a| = 25$ (five primitive steps of the five-dimensional arm actions), the refiner is $251 \to 512 \to 512 \to 125$, i.e. 0.46M parameters, applied with tied weights at all $K = 8$ refinement steps.

The RP1 actor is a weight-tied residual plan refiner. At refinement step k, it receives the current plan, its terminal value, and the value gradient with respect to the plan:

$$\hat{z}_N^{(k)} = H_\phi(\mathbf{a}_k, z_0), \tag{69}$$

$$v_k = V_{\bar{\psi}}(\hat{z}_N^{(k)}, z_g), \tag{70}$$

$$\mathbf{g}_k = \nabla_{\mathbf{a}_k} V_{\bar{\psi}}(\hat{z}_N^{(k)}, z_g). \tag{71}$$

The plan is updated by

$$\mathbf{a}_{k+1} = \mathrm{clip}_{[-a_{\max}, a_{\max}]} \left[\mathbf{a}_k + f_\theta(\mathbf{a}_k, v_k, \mathbf{g}_k) \right]. \tag{72}$$

The actor receives no raw current-state or goal latent. Goal information reaches it only through v_k and $\mathbf{g}_k$.

The actor is trained by differentiating the terminal value through the frozen world-model rollout. The value and gradient supplied as refiner inputs are detached, while the training loss remains differentiable through the refined action sequence and its resulting rollout. No environment interaction is used during this stage.

Let v_k be the terminal value after refinement step k. The planner objective is

$$J_{\mathrm{RP1}}(\theta) = \mathbb{E}_{\hat{z}^{(K)}} \Big[v_K + \lambda_{\mathrm{mean}} \frac{1}{K} \sum_{k=1}^{K} v_k \Big]. \tag{73}$$

The initial value from Section C.1 initializes the RP1 critic. When critic co-training is enabled, it continues to receive the same cached-data TD updates while a Polyak-averaged copy supplies v_k and $\mathbf{g}_k$.

All reported experiments execute each plan open loop before replanning. For closed-loop control, one could let $\hat{\tau}$ be the first imagined goal-arrival step, measured by $v_k \leq \epsilon$, or N if the goal is not reached, and optimize the telescoped per-step costs $\sum_{i=0}^{\hat{\tau}-1} \gamma^i (1 + \gamma V(\hat{z}^{(K)}_{t+i+1}, z_g) - V(\hat{z}^{(K)}_{t+i}, z_g))$, which yields an arrival-aware loss that favors reaching the goal earlier. We do not evaluate this variant.

C.3 Dyna Loop

The values v_k in the planner-loss are read off latents $H_\phi(\mathbf{a}_k, \hat{z}_0)$ that the world models h_ϕ produced. Should h_ϕ be wrong, or not have coverage for the dataset $\mathcal{D}$, the planner can exploit inaccuracies, as is well reported in literature [41; 47; 16].

Much of this can be fixed by finetuning the world model on actual roll-out data, as originally proposed in the Dyna loop[45]. For this we deploy θ_r in the real environment, collect the (failure) trajectories it produces, mix them into the training data, and finetune the world model on the mixture. Then we retrain the planner and repeat.

D EMPIRICAL RESULTS

D.1 GENERAL SETUP

Data and evaluation protocol. All world-model encoders and dynamics predictors are frozen throughout planner training. Each domain provides $10{,}000$ episodes: value functions and planners train on episodes 0–7,999, while all evaluations draw start/goal states from the held-out episodes 8,000–9,999. All model-selection decisions, including checkpoint selection and the stopping points in Table 5, are made exclusively on selection draws $\{48, 49, 50, 51\}$, which are never used for reported results. Selection evaluates 36 checkpoint pairs for six seeds on four draws of 50 episodes, i.e. $43{,}200$ simulator episodes per experimental cell. These episodes are used only to choose checkpoints; no gradient update uses environment data. Final reported numbers use the disjoint evaluation draws $\{42, 43, 44\}$, with 50 episodes per draw, and are the median over six planner training seeds $\{0, \ldots, 5\}$ of each seed's mean across those three evaluation draws.

Planning protocol. All planners use 5-step action chunks (25 primitive steps) and optimize them open loop, replanning every 5 chunks (receding horizon 5). The goal state is h primitive steps ahead and the episode budget has the length of $2h$ steps; TwoRoom and Cube evaluate $h \in \{25, 100\}$ ($h25$, $h100$), Reacher h=25. Simulator evaluations run under EGL with a pinned render device and at most one evaluation per GPU; we verified that this parallelism is bit-identical to fully serialized evaluation before adopting it.

Objectives. Every planner scores the predicted terminal state with one of the two objectives of Sec. D.5: the latent-distance objective $C_{\text{latent}}(\hat{z}_N, z_g) = \|\hat{z}_N - z_g\|_2^2$, or the value objective $C_{\text{value}}(\hat{z}_N, z_g) = V_\psi(\hat{z}_N, z_g)$, the goal-conditioned temporal-distance critic of Sec. C.1 (MRN quasimetric-style head) trained on the frozen cached latents of each base. All encoders and dynamics predictors remain frozen during critic and planner training, so differences in performance arise from how imagined trajectories are scored and improved rather than from changes to the world models.

Shared recipe across domains. Every RP1 result uses the same actor architecture, critic architecture, refiner architecture, optimization settings, and planner hyperparameters shown in Table 6 across all three environments and both world-model backbones. We make only two environment-independent adjustments dictated by the prediction horizon and observability. First, the critic discount is set by the horizon being evaluated, with γ=0.98 for $h25$ and γ=0.99 for $h100$, so that its effective horizon $1/(1-\gamma)$ covers the task horizon. Second, Reacher uses a two-frame latent cost representation for both the critic and latent-distance objective because joint velocity is otherwise unobservable from a single frame.

Stopping point. We train RP1 with early stopping. The offline critic is snapshotted every $3{,}000$ steps up to its $18{,}000$-step cap, and the planner every $2{,}000$ steps up to its $12{,}000$-step cap. Each of the 6×6 planner–critic checkpoint pairs is evaluated for all six training seeds on the four selection draws $\{48, 49, 50, 51\}$. For each experimental cell, we select a single checkpoint pair by its mean score across seeds and selection draws. Ties are broken toward the smaller training budget and then the earlier step. This selected pair is then fixed, and all six seeds are evaluated on the disjoint reporting draws $\{42, 43, 44\}$. No per-seed checkpoint selection is performed, and the same selection rule is applied to the learned baselines.

		25 steps		100 steps	
domain	base	critic	planner	critic	planner
TwoRoom	LeWM	15	2	6	8
	PLDM	3	2	3	10
Reacher	LeWM	18	2	—	—
	PLDM	18	2	—	—
Cube	LeWM	3	12	9	2
	PLDM	15	8	12	2

Table 5: **Selected stopping points**. The offline critic is snapshotted every 3k to its 18k cap and the planner every 2k to its 12k cap; one of the 6×6 pairs is fixed per cell as described above. Reacher is evaluated at $h25$ only. A planner entry of 12 is the step cap, i.e. the final iterate.

Hyperparameter	**Value**	**Hyperparameter**	**Value**
Plan refiner		**Co-trained critic**	
refinement iterations K	8	γ ($h25$ / $h100$)	0.98 / 0.99
plan horizon H (chunks)	5	n-step	1
replan every	5 chunks	expectile (annealed)	$0.1 \rightarrow 0.03$
clip range $a_{\max}$	2.5	critic LR (annealed)	$10^{-3} \rightarrow 10^{-4}$
refiner MLP (width $\times$ hidden layers)	512×2	live fraction of actor steps	0.8
		EMA τ / TD batch	0.005 / 1,024
Actor		head	MRN, $256/128$, depth 2
mean-weight λ_{mean}	0.3		
actor LR (cosine to $1/10$)	$3{\cdot}10^{-4}$	**Offline teacher**	
batch size	256	step cap / snapshot	18,000 / 3,000
step cap / snapshot	12,000 / 2,000	expectile	0.03
max-delta (chunks)	20	teachers per actor seed	1
cross-episode goal probability	0.3		
anti-constancy weight	0.5		

Table 6: **The RP1 configuration.**

D.2 TwoRoom

Setup. In TwoRoom, success is judged by whether the final distance to the goal is within 16 pixels. Figure 5 probes the learned critic, comparing latent distance to the critic's value landscape on sampled tasks; Fig. 6 traces plan refinement against the hand-designed planners.

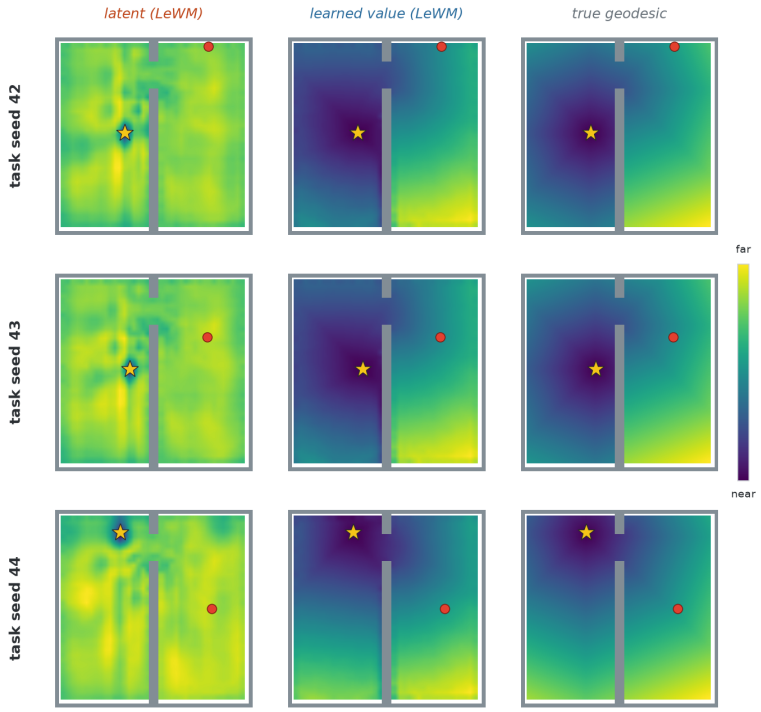


Figure 5: **Latent-Distance vs. learned Cost-to-go.** 3 randomly sampled tasks from seeds (42,43,44) and their corresponding cost landscapes.

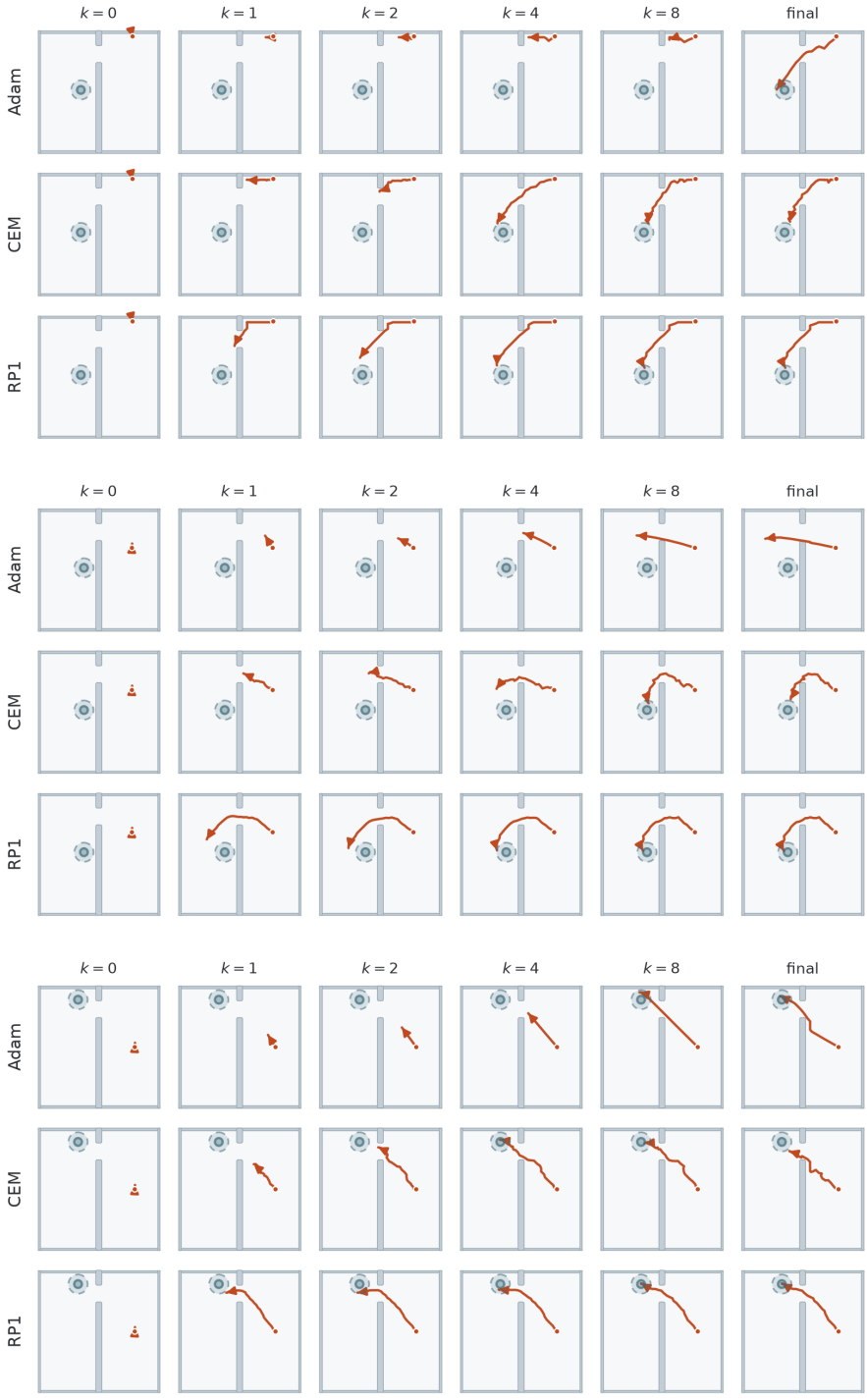


Figure 6: **Plan refinement in TwoRoom.** Each panel shows the planner's best-scoring candidate plan at refinement iteration k on the same task, with all planners scoring plans using the learned value critic; "final" is k=30 for Adam and CEM and k=8 for RP1. Iterations differ greatly in cost: one CEM iteration evaluates 300 sampled plans (9,000 rollouts in total), one Adam iteration takes a gradient step on 100 plans in parallel (3,000 rollouts), whereas one RP1 iteration is a single forward pass of the learned refiner costing one rollout (9 in total, including the initial evaluation).

D.3 REACHER

Setup. In Reacher success is first-hit: all joints within τ radians of the goal configuration, scored in a separate simulator pass per τ with termination on success.

Cost windows. On Reacher, both objectives read the predicted terminal state through a latent window of w terminal frames; we use w=1 for all reported results except Reacher, where we use $w = 2$, as joint velocity is not observable from a single frame, and a window of $w > 1$ lets the cost read it. The Reacher value critic additionally uses window lag 5 and standardized latents, and the RP1 co-trained critic is initialized from the offline value trained at the same cost window.

D.4 OGBENCH CUBE

Setup. Every cell is 50 episodes per evaluation seed. Success follows the benchmark's cube-placement criterion; the no-op floors (56.0 at $h25$, 45.3 at $h100$) are measured by executing zero actions under the identical protocol.

Dyna iterations. In each Dyna round, we collect on-policy episodes with the deployed planners of the previous round (all six seeds) on the training-split task. We collect at *both* evaluation horizons, 1,800 episodes at $h25$ and 1,800 at $h100$, and mix them 50:50 with the original expert data. The world model is then fine-tuned for one epoch at LR 10^{-5} from the previous round's model, with a latent anchor $\|\mathrm{enc}_{\mathrm{ft}}(x) - \mathrm{enc}_{\mathrm{prev}}(x)\|^2$ (weight 1.0) added to the objective on every frame of the mixture. RP1 is retrained at the cell's locked configuration, the teacher budget and actor step pair selected in Sec. D.1.

We run two rounds and report, per cell, the round with the better validation score: round 1 for LeWM (93.00 against round 2's 92.75) and round 2 for PLDM (90.33 against 83.25). We report Dyna at $h25$ only.

D.5 PLANNING BASELINES

Each conventional planner is evaluated with two terminal objectives. The latent-distance objective scores the predicted terminal latent by

$$C_{\mathrm{latent}}(\hat{z}_N, z_g) = \|\hat{z}_N - z_g\|_2^2. \tag{74}$$

The value objective uses the goal-conditioned value trained for the corresponding environment and world model:

$$C_{\mathrm{value}}(\hat{z}_N, z_g) = V_\psi(\hat{z}_N, z_g). \tag{75}$$

All baselines plan in the same normalized 5-chunk action space as RP1 and follow the identical receding-horizon protocol; they differ only in how the action sequence is optimized. Per decision, CEM and MPPI evaluate 9,000 forward rollouts; Adam evaluates 3,000 forward rollouts and the corresponding 3,000 backward passes.

Cross Entropy Method (CEM). CEM samples complete action sequences from a factorized Gaussian, retains the lowest-cost elite set, and refits the sampling distribution after every iteration. We use 300 samples per iteration for 30 iterations with an elite set of 30 (top 10%); the initial distribution is zero-mean with unit variance in the normalized action space.

Model-Predictive Path-Integral (MPPI). MPPI samples Gaussian perturbations around the current action sequence and updates the sequence using exponentially weighted trajectory costs. We use 300 samples per iteration for 30 iterations with temperature $\lambda = 0.5$.

Adam. Adam directly differentiates the terminal objective through the frozen world-model rollout and optimizes a batch of action sequences. We optimize 300 sequences in parallel for 10 steps with AdamW at learning rate 0.1 and execute the lowest-cost sequence. For TwoRoom we optimized 100 sequences in parallel at 30 steps.

Deep Model-Predictive Optimization (DMPO). DMPO learns a residual on the MPPI update: a network reads the sampling distribution and the N rollout costs and emits a gated mean correction, a covariance update, and a learned warm-start shift [39]. We train it with the original objective, PPO, inside the frozen world model against the same critic V_ψ used by RP1 in each cell, for the original budget of $1{,}000$ policy-improvement iterations, with six training seeds. Snapshots are taken every 200 iterations, and one shared snapshot per cell is chosen by the mean score over seeds on the selection draws, with ties broken toward the earlier snapshot. This is the same early-stopping rule used to select RP1's checkpoints. DMPO uses 256 rollouts per decision (256×1).

Learning-to-Optimize MPC (L2O-MPC). L2O-MPC learns to imitate the full sampling update of MPPI rather than a residual: a network reads the mean, covariance, and N costs and emits a gated replacement mean [38]. It is trained by DAgger imitation of a larger-budget MPPI expert, computed here through the same frozen world model and critic. We follow the original DAgger structure: 20 rounds, $\beta_k = 0.8^k$, and an expert budget well above the learner's ($N{=}512$ against $M{=}64$). The original optimizer budget of roughly 10^6 gradient steps per seed was not necessary, as we have found that training for $18{,}000$ steps already plateaued. We recorded snapshots every $3{,}000$ steps, and apply the same selection rule as for RP1. L2O-MPC uses 256 rollouts per decision (64×4), matching DMPO.