BEYOND ACTION PREDICTION: LEARNING SEARCH-FREE PLANNING IN LATENT SPACE

Anonymous authors
Paper under double-blind review

ABSTRACT

Latent world models learn action-conditioned state transitions in a compact representation space, providing a predictive model for visual planning and control. However, planning with these models typically requires costly test-time optimization over action sequences. We propose to replace this online search with a goal-conditioned inverse model that directly predicts actions from current and goal observations. A key challenge is that action prediction alone neither ensures that goal representations encode action-relevant information nor that predicted actions are consistent with the learned latent prediction model. We introduce Hindsight Inverse JEPA Planning (HIJP), a search-free framework that addresses both aspects during training. HIJP relabels future observations along offline trajectories as goals, providing action supervision across multiple temporal horizons. By retaining gradients through the goal representations, this supervision directly shapes the visual representation. HIJP further feeds predicted actions into the latent predictor model and aligns the resulting next-state representations with encoded successor observations, enforcing action-transition consistency. At inference time, the inverse dynamics model and latent predictor models are applied alternately to generate multi-step action sequences without online search. Across four visual control tasks, HIJP achieves 86.8% overall success and a 1.9–16.5× planning speedup over search-based planning.

1 INTRODUCTION

Latent world models encode high-dimensional visual observations into compact latent representations and learn an action-conditioned latent predictor from offline trajectories, providing a foundation for visual planning and control (Zhou et al., 2025; Maes et al., 2026; Zhang et al., 2025). However, planning with these models typically requires optimizers such as CEM (Rubinstein, 1999) or MPPI (Williams et al., 2017) to repeatedly generate and evaluate candidate action sequences at each replanning step, which requires substantial test-time computation (Pinneri et al., 2021). Learning a direct mapping from the current state and goal to actions therefore provides a promising alternative for improving the planning efficiency of latent world models (Sun et al., 2026; Nguyen et al., 2026).

Recent work has introduced action supervision to improve latent world models' control capabilities. SMWM learns action-relevant representations through inverse dynamics supervision (Ivashkov et al., 2026), while Delta-JEPA (Zhang et al., 2026) recovers actions from latent state differences, encouraging representations to retain action-related state changes. GC-IDM (Nguyen et al., 2026) further learns a goal-conditioned action mapping on frozen representations, while INTACT (Sun et al., 2026)incorporates local and future-goal action supervision into end-to-end training, supporting search-free recursive control. These approaches demonstrate the value of action supervision for goal-conditioned control. However, accurate action prediction alone does not necessarily provide all the supervision needed for search-free planning.

We identify two aspects that remain underconstrained. First, goal-conditioned action supervision should not only train an inverse model to predict actions, but also help shape visual representations to capture how actions relate current states to future goals. Offline trajectories naturally contain such relations at multiple temporal scales: for the same current state and demonstrated action, different future observations define goals at different temporal horizons. However, existing training schemes (Sun et al., 2026; Wang et al., 2026a) exploit only a subset of these relations, for exam-

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.

ple by conditioning action prediction on a fixed future endpoint. More generally, treating the goal representation only as a conditioning target limits the extent to which action supervision can shape goal-side visual features. Second, accurate action prediction alone does not ensure that the predicted action induces the correct transition under the learned latent predictor. The inverse dynamics model and the latent predictor model are typically supervised separately, with the latter conditioned on demonstrated actions (Ivashkov et al., 2026; Sun et al., 2026). During search-free rollout planning, however, these two mappings are applied sequentially: the inverse dynamics model predicts an action toward the goal, and the latent predictor model uses this predicted action to advance the latent state for subsequent action generation. The resulting inverse-to-predictor composition is therefore directly involved in planning, yet is not explicitly constrained by separate action and latent prediction objectives. This motivates extending action supervision beyond action prediction itself, using it both to shape goal-conditioned representations and to couple predicted actions with their induced latent transitions.

Based on this principle, we propose Hindsight Inverse JEPA Planning (HIJP), which jointly trains a visual encoder, a goal-conditioned inverse dynamics model, and a latent predictor model end-to-end. HIJP relabels future observations within each trajectory window as goals, exposing the inverse objective to diverse state–goal relations across temporal horizons. Gradients are retained through the goal representations, allowing inverse-action supervision to directly shape the visual features used for goal-conditioned action prediction. We further feed the predicted actions into the latent predictor and align the resulting successor latents with the encodings of the next observations. This action-transition consistency objective jointly updates the inverse and predictor models, directly coupling action generation with its induced latent transition. In this way, HIJP trains the same inverse-to-predictor mapping that is repeatedly applied during planning. At test time, the inverse dynamics module and latent predictor model alternate to generate multi-step action sequences, which are executed through closed-loop replanning without online action search.

We evaluate HIJP on four visual control tasks under four configurations of goal temporal offsets and execution budgets. HIJP achieves an overall success rate of 86.8%, outperforming the strongest baseline by 4.9 percentage points. Ablations on hindsight goal sampling, training objectives, and gradient routing examine how goal-side representation learning and inverse - predictor coupling contribute to control performance. Beyond success rates, we also evaluate efficiency and the temporal smoothness of executed actions. With matched numbers of replanning calls, HIJP achieves a 1.9–$16.5\times$ planning speedup over LeWM. Under the most challenging setting, its executed actions exhibit lower jerk than those produced by CEM using the same representations, indicating smoother executed controls.

2 Related work

Latent World Models Latent world models learn action-conditioned changes in representation space to support goal-directed planning from offline visual trajectories. Existing approaches span pretrained visual features, end-to-end representation learning, and large-scale video pretraining (Zhou et al., 2025; Maes et al., 2026; Assran et al., 2025). Systematic studies further investigate how architectures, training objectives, and planning algorithms affect control performance (Terver et al., 2025). Gao & Xu (2026) reduce forward-rollout costs through parallel prediction conditioned on action prefixes. Wang et al. (2026a) use inverse dynamics and goal-conditioned action supervision to improve planning representations, while retaining CEM for control. HIJP builds on latent forward dynamics and focuses on learning a goal-conditioned action mapping that can be directly deployed for control.

Inverse Dynamics and Goal-Conditioned Action Learning Inverse dynamics and action-supervised objectives encourage representations to retain action-relevant state changes. These objectives can also be combined with physical-state alignment to improve representations for control (Ivashkov et al., 2026; Zhang et al., 2026; Liu et al., 2026). Hindsight learning relabels achieved states along trajectories as goals, providing supervision for goal-conditioned policies and inverse dynamics learning (Andrychowicz et al., 2017; Ghosh et al., 2019; Sun et al., 2019). Goal-conditioned action-sequence prediction has also been used for visual planning (Paster et al., 2020). Related theoretical studies analyze the benefits of inverse dynamics through bias–variance trade-offs and learning-problem complexity (Schäfer et al., 2026; Morin et al., 2026). Within latent world models,

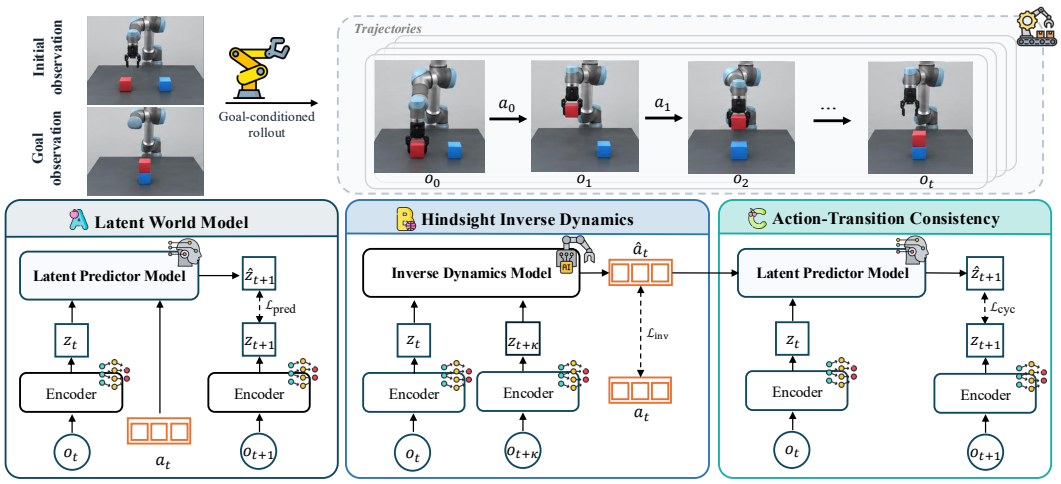

Figure 1: Overview of the HIJP training framework. (A) The latent world model predicts the next latent state from the current state and action. (B) The hindsight-conditioned inverse dynamics model infers the action from current and future latent states. (C) The inferred action is reused to enforce action–transition consistency. All modules are jointly optimized using the prediction, regularization, inverse-dynamics, and action transition consistency losses.

GC-IDM(Nguyen et al., 2026) learns a goal- and horizon-conditioned action mapping on frozen representations, while INTACT (Sun et al., 2026) jointly learns local and goal-conditioned action prediction to support search-free recursive control. HIJP follows this direction by enriching supervision through future-goal relabeling within trajectory windows.

Learned and Amortized Planning Planning computation can be shifted to training by learning action priors, latent subgoals, or plan-update rules (Wang et al., 2026b; Cheng et al., 2026; Masip et al., 2026; Sommer & Schilling, 2026). Another line of work first predicts future states or generates state trajectories and then recovers actions through inverse dynamics (Tian et al., 2025; Xie et al., 2025; Ajay et al., 2022; Huang et al., 2026). Rakhimov et al. (2026) jointly train state and action flows, allowing a single model to support multiple control inference modes. HIJP alternates goal-conditioned action prediction with forward latent-state prediction to construct multi-step plans without online action search.

Consistency and Goal-Aware Planning Objectives Consistency objectives also connect model prediction with planning. Seo et al. (2026) incorporate action-recovery errors from predicted transitions into the online planning cost, while Bai & Xiong (2026) combine temporal-progress learning with multi-step rollout consistency supervision. HIJP focuses on the composed output of the goal-conditioned action module and the forward model. During training, it aligns the successor latent state predicted under the generated action with the encoding of the next observation in the recorded trajectory, allowing state-prediction errors to supervise action learning directly.

3 METHODOLOGY

Our goal is to extend action supervision beyond action prediction itself, such that it also shapes goal-conditioned visual representations and constrains the latent transitions induced by predicted actions. HIJP jointly learns a visual encoder f_θ, a latent predictor model g_ϕ, and a goal-conditioned inverse dynamics model h_ξ. Given offline visual trajectories, the inverse dynamics model learns to predict actions toward future goals, while the predictor predicts action-conditioned latent transitions. We couple these components during training to supervise the local inverse-to-predictor composition repeatedly used in search-free planning. Figure 1 provides an overview.

3.1 Problem Formulation

Let $\mathcal{D} = \{\tau^{(i)}\}_{i=1}^{N}$ be a dataset of trajectories, where

$$\tau = (o_0, \mathbf{a}_0, o_1, \ldots, \mathbf{a}_{T-1}, o_T). \tag{1}$$

Here o_t is a visual observation and $\mathbf{a}_t$ is the action taken at time t. Given a current observation o_t and a goal observation o_g, the task is to produce actions that reach the goal within a fixed interaction budget. Training uses recorded observations and actions, with future observations in each trajectory serving as hindsight goals, where reward annotations are not required. Our planning procedure repeatedly composes two mappings. An inverse model predicts an action from the current latent state and goal latent state, and a latent predictor model predicts the latent state induced by that action. This composition motivates training the models together: action prediction must identify a useful action, while the resulting transition must agree with the observed dynamics.

3.2 Latent World Model

Following the prior work (Maes et al., 2026), a visual encoder $f_\theta : \mathcal{O} \to \mathbb{R}^d$ maps observations to latent states $z_t = f_\theta(o_t)$, and an action-conditioned latent predictor model g_ϕ predicts the next latent state from a causal context of latent states and actions:

$$\hat{z}_{t+1} = g_\phi(z_{t-m+1:t}, a_{t-m+1:t}), \qquad z_{t+1} = f_\theta(o_{t+1}). \tag{2}$$

We train the latent prediction model on demonstrated transitions using

$$\mathcal{L}_{\text{pred}} = \text{MSE}(\hat{z}_{t+1}, z_{t+1}). \tag{3}$$

We also apply $\mathcal{L}_{\text{sigreg}}$ to the encoded observations in each training window to discourage representational collapse. We use a ViT-Tiny encoder trained from scratch and a causal Transformer model with context length $m = 3$. An action encoder supplies the latent prediction model with action embeddings. Architecture and optimization details are given in Appendix B. The same model also accepts the shorter context used for the one-step consistency objective below.

The latent prediction model learns how demonstrated actions transform latent states. Our next objective is complementary: to learn how actions can be inferred from current and goal representations, while allowing this action supervision to shape the representations themselves.

3.3 Hindsight-conditioned Inverse Dynamics

A trajectory contains multiple valid future goals for a given observation. To expose these relations during training, we sample a window of W observations, an anchor i and a future offset κ:

$$i \sim \text{Unif}\{0, \ldots, W-2\}, \qquad \kappa \mid i \sim \text{Unif}\{1, \ldots, W-1-i\}. \tag{4}$$

Each sampled pair therefore defines a goal-conditioned action-learning problem at a potentially different temporal horizon. Unlike using only the endpoint of a trajectory window, this construction exposes the inverse objective to diverse relations between current observations and future goals using the same offline data. The observation at $i + \kappa$ becomes the hindsight goal. The goal-conditioned inverse model predicts the demonstrated action at the anchor:

$$\hat{a}_i = h_\xi(z_i, z_{i+\kappa}), \qquad \mathcal{L}_{\text{inv}} = \text{MAE}(\hat{a}_i, a_i). \tag{5}$$

Although the sampled goal may be several steps ahead, the target is the immediate action a_i, not the intervening action sequence. Each training window therefore supplies action supervision under goals at different temporal offsets.

Gradients from $\mathcal{L}_{\text{inv}}$ flow through both z_i and $z_{i+\kappa}$. Consequently, the inverse objective trains not only h_ξ, but also the visual features used to represent the current observation and the future goal. We implement h_ξ as an MLP operating on the concatenated latent pair.

3.4 Action Transition Consistency

The inverse model can predict an action close to the demonstrated action without ensuring that this prediction produces the observed transition under the learned predictor model. This distinction

matters at planning time, where the model receives predicted rather than demonstrated actions. We therefore pass the predicted action through g_ϕ:

$$\tilde{z}_{i+1} = g_\phi(z_i, \hat{a}_i)\,, \qquad \hat{a}_i = h_\xi(z_i, z_{i+\kappa}), \tag{6}$$

and align the result with the encoded one-step successor:

$$\mathcal{L}_{\text{cyc}} = \text{MSE}(\tilde{z}_{i+1}, \text{sg}[z_{i+1}])\,, \tag{7}$$

where sg denotes stop-gradient. This objective uses the one-step successor z_{i+1} even when the hindsight goal $z_{i+\kappa}$ is farther away. It allows L_{cyc} to update the inverse head, action encoder, latent prediction model, and the anchor-side visual representation. Thus, the consistency objective does not merely use the prediction model to provide an error signal to the inverse dynamics model; it jointly adapts action generation and the latent predictor around their composed prediction. This directly supervises the inverse-to-latent predictor transition that is repeatedly composed during search-free planning.

3.5 Joint Objective and Search-free Planning

Joint training. The full training objective combines latent prediction, representation regularization, hindsight-conditioned action prediction, and action-transition consistency:

$$\mathcal{L} = \mathcal{L}_{\text{pred}} + \lambda_{\text{reg}}\,\mathcal{L}_{\text{sigreg}} + \lambda_{\text{inv}}\,\mathcal{L}_{\text{inv}} + \lambda_{\text{cyc}}\,\mathcal{L}_{\text{cyc}}. \tag{8}$$

All components are optimized jointly, and all terms are averaged over their corresponding batch, temporal, sampled-pair, and feature dimensions before weighting. For all tasks, the loss weights are set to $\lambda_{\text{reg}} = 0.09$, $\lambda_{\text{inv}} = 1.0$, and $\lambda_{\text{cyc}} = 0.05$ for TwoRoom and 0.10 for the remaining tasks.

Search-free planning. At planning time, we use the inverse dynamics model and latent prediction models in the same order in which they are coupled during training. At each replanning point, the latest observation and fixed goal observation are encoded as $z_t = f_\theta(o_t)$ and $z_g = f_\theta(o_g)$. Starting from $\hat{z}_0 = z_t$, we construct an H-step plan by alternating goal-conditioned action prediction and latent prediction:

$$a_j = h_\xi(\hat{z}_j, z_g), \qquad \hat{z}_{j+1} = g_\phi\big(\hat{z}_{\ell_j:j}, a_{\ell_j:j}\big)\,, \quad \ell_j = \max(0, j-m+1), \quad j = 0, \ldots, H-1. \tag{9}$$

The inverse model therefore predicts each action from the current imagined state and fixed goal, while the latent prediction model advances the imagined state under that predicted action. We use $H = 5$, execute the resulting action sequence, observe the environment again, and repeat the procedure in a receding-horizon manner. Each replanning call requires only one inverse-to-latent prediction rollout, without sampling or iteratively refining candidate action sequences.

4 Experiments

We conduct experiments on four tasks to answer the following key questions: (i) whether our method achieves strong goal-reaching performance, particularly for long-horizon goals; (ii) how much inference speed is gained by replacing the CEM search at test time with a single amortized rollout; (iii) how hindsight goal relabeling, inverse-action supervision, gradient routing, and predicted-action transition consistency contribute to control performance; and (iv) how HIJP and CEM differ in latent-space goal progress and executed-action smoothness.

4.1 Experiment setup

Benchmarks and Tasks We evaluate on four goal-conditioned control benchmarks with distinct dynamics: *TwoRoom* (2D navigation), *Reacher* (DMControl reacher), *PushT* (planar pushing), and *OGCube* (OGBench cube manipulation). All tasks use 224×224 RGB observations, continuous actions, and observation-specified goals. The dataset information is provided in Appendix A. We use the offline datasets released by LeWM (Maes et al., 2026) without modification. To vary difficulty, we set each goal to the environment state $K \in \{10, 25, 50, 75\}$ steps ahead of the start and assign a corresponding interaction budget $B \in \{25, 50, 100, 100\}$, defining the *easy*, *default* (LeWM's(Maes et al., 2026) original goal-setting horizon and interaction budget), *hard*, and *extreme* tiers, respectively. All methods are evaluated in the original simulator on the same start–goal instances, with 50 episodes per task–tier combination. An episode counts as successful only when the simulator's native success condition is met; we use no latent-space distance threshold. We report success rate as the primary metric and planning time for the efficiency study, with additional details in Appendix A.2.

Table 1: Success rates (%) across four tasks and four difficulty tiers. Avg. denotes the arithmetic mean across the four difficulty tiers; Overall Avg. averages all 16 task–tier combinations with equal weight. The best and second-best results in each row are shown in **bold** and underlined text.

		Reference	Baselines							Ours
Task	Tier	Random	GCBC	PLDM	DINO-WM	LeWM	SMWM	Δ-JEPA	INTACT	**HIJP**
TwoRoom	Easy	66%	**100%**	**100%**	**100%**	96%	96%	76%	**100%**	**100%**
	Default	24%	**100%**	88%	**100%**	88%	96%	38%	**100%**	**100%**
	Hard	8%	**100%**	90%	98%	62%	86%	48%	96%	**100%**
	Extreme	2%	**100%**	96%	74%	50%	72%	24%	90%	98%
	Avg.	25.0%	**100.0%**	93.5%	93.0%	74.0%	87.5%	46.5%	96.5%	99.5%
Reacher	Easy	12%	4%	**70%**	46%	40%	34%	14%	44%	48%
	Default	10%	0%	72%	78%	76%	46%	20%	**100%**	92%
	Hard	6%	0%	72%	98%	96%	62%	52%	**100%**	92%
	Extreme	6%	0%	84%	94%	84%	58%	40%	**96%**	94%
	Avg.	8.5%	1.0%	74.5%	79.0%	74.0%	50.0%	31.5%	**85.0%**	81.5%
PushT	Easy	14%	94%	94%	**98%**	78%	80%	72%	**98%**	90%
	Default	2%	74%	42%	74%	**88%**	56%	64%	80%	80%
	Hard	2%	40%	22%	56%	58%	30%	36%	54%	**66%**
	Extreme	0%	4%	0%	28%	12%	16%	0%	6%	**42%**
	Avg.	4.5%	53.0%	39.5%	64.0%	59.0%	45.5%	43.0%	59.5%	**69.5%**
OGCube	Easy	68%	98%	74%	92%	98%	98%	88%	**100%**	98%
	Default	44%	90%	56%	78%	82%	88%	74%	**100%**	96%
	Hard	26%	58%	28%	62%	56%	70%	54%	70%	**96%**
	Extreme	40%	76%	50%	72%	70%	86%	74%	76%	**96%**
	Avg.	44.5%	80.5%	52.0%	76.0%	76.5%	85.5%	72.5%	86.5%	**96.5%**
Overall Avg.		20.6%	58.6%	64.9%	78.0%	70.9%	67.1%	48.4%	81.9%	**86.8%**

Baselines We compare against eight baselines: Random, GCBC (Ghosh et al., 2021), DINO-WM (Zhou et al., 2025), PLDM (Sobal et al., 2025), LeWM (Maes et al., 2026), SMWM (Ivashkov et al., 2026), delta-JEPA (Zhang et al., 2026) and INTACT(Sun et al., 2026). Random provides a chance-level reference; GCBC and INTACT provide direct goal-conditioned controllers, with INTACT representing the closest end-to-end search-free baseline. The remaining methods are latent world-model planners that rely on CEM- or MPPI-based search at test time. All learned methods are trained on the same offline datasets and evaluated under the same protocol. All CEM-based comparisons use the shared search and execution protocol specified in Appendix A.3.

Implementation Details HIJP combines a latent world model with an amortized inverse head trained end-to-end. At planning time, it produces an $H = 5$ step latent plan with a single amortized rollout, executes the corresponding actions in a receding-horizon manner, and then replans from the latest observation. Unlike the CEM-based world-model baselines, HIJP uses no test-time search. Architecture and optimization details are provided in Appendix B.

4.2 Planning Performance

We first examine how HIJP's goal-reaching performance compares with the baselines. As detailed in Section 4.1, we test HIJP and all baseline methods on each task-tier combination, using 50 episodes per pair, and report success rates in Table 1.

Table 1 shows that HIJP achieves the highest overall success rate of 86.8%, exceeding the strongest baseline by 4.9 percentage points. The gains are most pronounced in challenging manipulation settings. On PushT, HIJP achieves 66% and 42% success in the hard and extreme tiers, respectively, outperforming the best baseline in each tier by 8 and 14 percentage points. On OGCube, it maintains 96% success in both tiers, with corresponding gains of 26 and 10 percentage points. On TwoRoom and Reacher, HIJP remains competitive, ranking second in average success on both tasks. The stronger gains on manipulation are consistent with our design to extend action supervision beyond action prediction itself, shaping goal-conditioned visual representations and constraining the latent

Table 2: Total wall-clock planning time (seconds) for LeWM and HIJP. Speedup denotes LeWM's planning time divided by HIJP's. Avg. reports the arithmetic mean across difficulty tiers, including the per-tier speedup ratios. Lower planning times are shown in **bold**.

Task	Method	Difficulty tier				Avg.
		Easy	Default	Hard	Extreme	
TwoRoom	LeWM	69.8	124.9	254.7	253.9	175.9
	HIJP	**13.5**	**20.8**	**36.2**	**37.4**	**27.0**
	Speedup	5.2×	6.0×	7.0×	6.8×	6.2×
Reacher	LeWM	153.6	296.0	566.6	581.9	399.5
	HIJP	**32.6**	**45.4**	**65.3**	**69.0**	**53.1**
	Speedup	4.7×	6.5×	8.7×	8.4×	7.1×
PushT	LeWM	183.0	368.5	682.3	669.6	475.8
	HIJP	**14.9**	**24.3**	**41.4**	**41.6**	**30.6**
	Speedup	12.3×	15.2×	16.5×	16.1×	15.0×
OGCube	LeWM	110.5	180.9	298.2	297.7	221.8
	HIJP	**59.5**	**70.5**	**87.4**	**90.8**	**77.0**
	Speedup	1.9×	2.6×	3.4×	3.3×	2.8×

transitions induced by predicted actions. We examine the contributions of these two design choices through the ablations in Section 4.4.

4.3 Planning Efficiency

To assess planning efficiency, we compare HIJP with LeWM, a CEM-based world-model planner, under matched numbers of replanning calls. Following Section 4.1, we report total wall-clock planning time for each task–tier combination, with all episodes processed as a single vectorized batch on an RTX 4090 (Table 2).

The result shows that HIJP reduces planning time across all evaluated settings, achieving 1.9–$16.5\times$ speedups over LeWM under matched replanning counts. The average speedups are $6.2\times$, $7.1\times$, $15.0\times$, and $2.8\times$ on TwoRoom, Reacher, PushT, and OGCube, respectively. Together with the results in Table 1, these findings show that HIJP achieves substantial planning speedups over LeWM while attaining higher average success rates on all four tasks.

4.4 Ablations

To systematically evaluate the contribution of each component, we conduct ablation studies on the training objectives, gradient propagation, hindsight goal relabeling, and the test-time planner. For the training objectives, we remove one term at a time from Eq. 8: the inverse loss $\mathcal{L}_{\text{inv}}$, the transition-consistency loss $\mathcal{L}_{\text{cyc}}$, and the SIGReg regularizer $\mathcal{L}_{\text{sigreg}}$.

We further evaluate two gradient-related variants. In $\text{sg}[z_g]$, we stop gradients through the goal representation z_g. In the frozen-g_ϕ variant, we retain the latent predictor g_ϕ in the forward computation but block parameter updates from $\mathcal{L}\text{cyc}$ while retaining those from $\mathcal{L}\text{pred}$. To evaluate the contribution of hindsight goal relabeling, we introduce a fixed-goal variant that always uses the latent representation of the final observation in each training window as the goal. Specifically, we sample an anchor step t uniformly from $\{0, \ldots, T-2\}$, set $z_g = z_{T-1}$, and predict the one-step action a_t from (z_t, z_g).

For the planner ablation, we keep the fully trained representation fixed and replace our amortized rollout with CEM search, thereby isolating the effect of the planning mechanism. All other settings follow Section 4.1. We report results on Reacher, PushT, and OGCube in Table 3. We exclude TwoRoom because all variants achieve near-100% success rates on this task. Furthermore, we examine sensitivity to the inverse and transition-consistency loss weights in Appendix C.

The results show that removing $\mathcal{L}_{\text{inv}}$ causes the largest overall decline, reducing average success from 82.5% to 45.7%. Removing SIGReg also reduces average success substantially, to 47.0%,

Table 3: Success rates for HIJP and its ablation variants. Avg. reports the arithmetic mean across the four difficulty tiers for each task, and Overall Avg. averages all 12 task–tier combinations. The best and second-best results in each row are shown in **bold** and underlined text.

		Full model	Training objectives		Goal supervision		Action Consistency		Planner
Task	Tier	**HIJP**	w/o $\mathcal{L}_{\text{inv}}$	w/o $\mathcal{L}_{\text{sigreg}}$	Endpoint goal	$\text{sg}[z_g]$	w/o $\mathcal{L}_{\text{cyc}}$	Frozen g_ϕ	CEM
Reacher	Easy	**48%**	36%	2%	36%	40%	34%	40%	40%
	Default	**92%**	72%	0%	66%	88%	78%	84%	70%
	Hard	92%	90%	0%	72%	88%	84%	80%	**98%**
	Extreme	**94%**	86%	0%	82%	92%	**94%**	84%	**94%**
	Avg.	**81.5%**	71.0%	0.5%	64.0%	77.0%	72.5%	72.0%	75.5%
PushT	Easy	**90%**	36%	74%	76%	**90%**	84%	86%	78%
	Default	**80%**	6%	58%	74%	**80%**	78%	72%	74%
	Hard	**66%**	0%	34%	50%	48%	54%	64%	44%
	Extreme	42%	0%	20%	44%	38%	22%	**46%**	24%
	Avg.	**69.5%**	10.5%	46.5%	61.0%	64.0%	59.5%	67.0%	55.0%
OGCube	Easy	**98%**	80%	**98%**	92%	94%	**98%**	**98%**	96%
	Default	**96%**	56%	**96%**	88%	**96%**	**96%**	**96%**	80%
	Hard	**96%**	46%	90%	88%	94%	94%	94%	58%
	Extreme	**96%**	40%	92%	92%	94%	**96%**	**96%**	76%
	Avg.	**96.5%**	55.5%	94.0%	90.0%	94.5%	96.0%	96.0%	77.5%
Overall Avg.		**82.5%**	45.7%	47.0%	71.7%	78.5%	76.0%	78.3%	69.3%

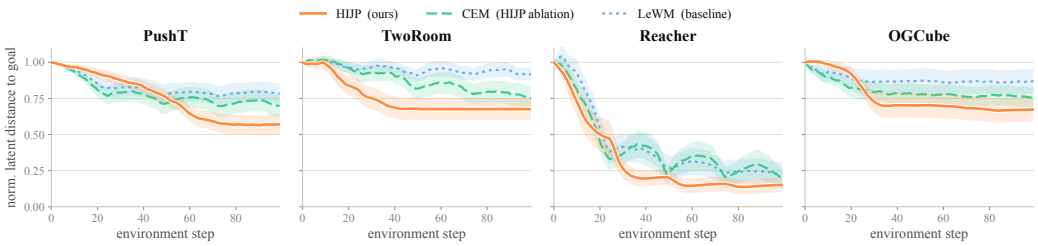

Figure 2: Normalized latent distance to the goal for HIJP, its CEM ablation, and LeWM across four tasks on the extreme tier. Lines show means, and shaded regions indicate pointwise 95% confidence intervals. HIJP and its CEM ablation share the same latent space, while LeWM uses its own encoder.

with near-collapse on Reacher. These results support the importance of action supervision and representation regularization for HIJP's control performance. The two goal-supervision ablations examine different parts of our design choice. Restricting goals to the final observation of each window reduces average success to 71.7%, while stopping gradients through z_g reduces it to 78.5%. This supports using multiple future observations as goals and allowing action-related gradients to shape goal representations.

The action consistency ablations examine the contribution of the transition-consistency objective. Removing $\mathcal{L}_{\text{cyc}}$ lowers average success to 76.0%, with the clearest decreases on Reacher and PushT and little change on OGCube. Blocking predictor updates from $\mathcal{L}_{\text{cyc}}$ lowers average success to 78.3%, primarily affecting Reacher. These results support the transition-consistency objective and its updates to the latent predictor, with gains varying across tasks. Finally, replacing the inverse-guided rollout with CEM reduces average success to 69.3% under the matched evaluation protocol, supporting the effectiveness of HIJP's learned planning mechanism.

4.5 Latent-Space Progress Analysis

HIJP shows its largest gains on long-horizon goals. We next examine how the latent distance to the goal evolves during execution. We compare HIJP, its CEM ablation (Section 4.4), and LeWM. The CEM ablation shares HIJP's learned representation and replaces HIJP's amortized inverse-guided rollout with CEM search, isolating the contribution of the planner. On the extreme tier (K=75), we execute all methods in the original simulator on the same start–goal instances, following each method's replanning schedule, and track the normalized latent distance to the goal, $d_t = \|z_t -$

$z_g\|/\|z_0 - z_g\|$, where z_t and z_g are the encoded observation and goal. After an episode succeeds, we hold its last recorded distance constant for the remaining steps. Figure 2 reports the mean and pointwise 95% confidence intervals.

Figure 2 shows that HIJP reaches a lower final normalized latent distance than its CEM ablation on all four tasks. On PushT, the methods exhibit similar initial progress, but HIJP continues to reduce the distance later in the episode as CEM largely plateaus. On TwoRoom and OGCube, HIJP reaches a lower plateau, while on Reacher it exhibits fewer fluctuations after the initial decrease. Compared with its CEM ablation, HIJP's executed trajectories end closer to the goal in the shared latent representation.

4.6 Smoothness of Executed Actions

We additionally evaluate the temporal smoothness of executed actions. We quantify this with action jerk, the mean ℓ_2 norm of the second finite difference of the executed action sequence, $\frac{1}{T-2}\sum_t \|\mathbf{a}_t - 2\mathbf{a}_{t-1} + \mathbf{a}_{t-2}\|_2$ (lower is smoother), following the convention of Nguyen et al. (2026). To attribute any difference to the planner rather than the representation, we compare HIJP against its CEM ablation, which shares the same learned representation and differs only in the planning rule, and include LeWM as a reference. Table 4 reports the extreme tier (K=75); the metric definition and full per-tier results are in Appendix D.

Table 4: Action jerk on the extreme tier. Ratio denotes each comparator's jerk divided by HIJP's. The lowest jerk in each environment is shown in **bold**.

	HIJP	CEM (ablation)		LeWM	
Environment	Jerk ↓	Jerk ↓	Ratio ↑	Jerk ↓	Ratio ↑
PushT	**0.048**	0.697	14.7×	0.724	15.2×
TwoRoom	**0.146**	2.503	17.1×	2.558	17.5×
Reacher	**0.046**	1.222	26.4×	1.134	24.5×
OGCube	**0.099**	2.065	20.9×	2.304	23.3×

The results show that HIJP produces smoother executed actions across all four tasks, with action jerk 14.7–26.4× lower than its CEM ablation and 15.2–24.5× lower than LeWM. These results indicate that HIJP's amortized rollout improves the temporal smoothness of executed controls in addition to its goal-reaching performance and planning efficiency.

5 Conclusion

We introduced Hindsight Inverse JEPA Planning (HIJP), a search-free framework for goal-conditioned visual control. HIJP samples future observations as hindsight goals to enrich inverse-dynamics supervision and allows training gradients to update goal representations. It further feeds predicted actions into a latent predictor model and constrains the resulting latent transitions through a consistency objective. At planning time, the inverse and latent predictor models alternate to construct multi-step action plans without online search. Across four visual control tasks and multiple evaluation configurations, HIJP achieves the highest aggregate goal-reaching performance among the evaluated methods while reducing planning time compared to a search-based world-model planner. Under the most challenging setting, its executed actions are also smoother than those produced by CEM using the same representations. Together, these results suggest that aligning goal-conditioned action learning with the latent transitions used during planning provides an effective route toward efficient, search-free visual control.

6 Limitations

Our evaluation focuses on tasks with a limited temporal span, using fixed planning horizons and replanning schedules. Extending HIJP to substantially longer-horizon tasks remains an important direction. Future work will explore hierarchical subgoal generation, adaptive planning horizons, and dynamic replanning schedules. These extensions aim to decompose longer tasks into shorter goal-conditioned segments and adapt planning and execution intervals to the requirements of each segment.

STATEMENT

REPRODUCIBILITY STATEMENT

HIJP's training and planning procedures are summarized in Algorithms 1 and 2 in the Appendix. Appendix A describes the datasets and evaluation protocol, while Appendix B provides the model architectures, training hyperparameters, and planning settings. Additional experimental details and results are provided in the remaining appendices. We will release the code and experiment configurations with the final version.

AI USE STATEMENT

We used a large language model to improve the grammar and readability of this manuscript. The authors take responsibility for the final content of this work, including all AI-assisted text.

REFERENCES

Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba, Komeili, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, Sergio Arnaud, Abha Gejji, Ada Martin, Francois Robert Hogan, Daniel Dugas, Piotr Bojanowski, Vasil Khalidov, Patrick Labatut, Francisco Massa, Marc Szafraniec, Kapil Krishnakumar, Yong Li, Xiaodong Ma, Sarath Chandar, Franziska Meier, Yann LeCun, Michael Rabbat, and Nicolas Ballas. V-jepa 2: Self-supervised video models enable understanding, prediction and planning, 2025. URL https://arxiv.org/abs/2506.09985.

Jiaxin Bai and Jiaxuan Xiong. Temporal-distance jepa: Plan-aware representation learning for latent world model predictive control. *arXiv preprint arXiv:2607.25337*, 2026.

Randall Balestriero and Yann LeCun. Lejepa: Provable and scalable self-supervised learning without the heuristics. *arXiv preprint arXiv:2511.08544*, 2025.

Letian Cheng, Qi Zhang, and Yisen Wang. Sage: Subgoal-conditioned action generation for latent world model planning. *arXiv preprint arXiv:2607.17973*, 2026.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Yuntian Gao and Xiangyu Xu. Fast leworldmodel. *arXiv preprint arXiv:2606.26217*, 2026.

Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. *arXiv preprint arXiv:1912.06088*, 2019.

Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Manon Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=rALA0Xo6yNJ.

Hsiang-Wei Huang, Jianxu Shangguan, Junbin Lu, and Jenq-Neng Hwang. Leflow: Generative latent flow planning for world models. *arXiv preprint arXiv:2608.24855*, 2026.

Petr Ivashkov, Randall Balestriero, and Bernhard Schölkopf. Sensorimotor world models: Perception for action via inverse dynamics, 2026. URL https://arxiv.org/abs/2606.20104.

Muyuan Liu, Yue Huang, Zheng Liang, and Xiang Gao. Toward physically grounded jepa world models for goal-conditioned robotic planning, 2026. URL https://arxiv.org/abs/2609.03565.

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.

Sergi Masip, Jonathan Swinnen, Yutong Hu, Renaud Detry, and Tinne Tuytelaars. Ff-jepa: Long-horizon planning in world models with latent planners. *arXiv preprint arXiv:2606.09311*, 2026.

Sacha Morin, Moonsub Byeon, Alexia Jolicoeur-Martineau, and Sébastien Lachapelle. On the sample efficiency of inverse dynamics models for semi-supervised imitation learning. *arXiv preprint arXiv:2602.02762*, 2026.

Hoang Nguyen, Xiaohao Xu, and Xiaonan Huang. Latent geometry beyond search: Amortizing planning in world models, 2026. URL https://arxiv.org/abs/2605.08732.

Keiran Paster, Sheila A McIlraith, and Jimmy Ba. Planning from pixels using inverse dynamics models. *arXiv preprint arXiv:2012.02419*, 2020.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4195–4205, October 2023.

Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolinek, and Georg Martius. Sample-efficient cross-entropy method for real-time planning. In *Conference on Robot Learning*, pp. 1049–1065. PMLR, 2021.

Ruslan Rakhimov, George Bredis, Yuriy Maksyuta, and Daniil Gavrilov. Qantara: Bridge-flow training for multi-paradigm jepa control. *arXiv preprint arXiv:2607.04978*, 2026.

Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1(2):127–190, 1999.

Lukas Schäfer, Pallavi Choudhury, Abdelhak Lemkhenter, Chris Lovett, Somjit Nath, Luis França, Matheus Ribeiro Furtado de Mendonça, Alex Lamb, Riashat Islam, Siddhartha Sen, et al. When does predictive inverse dynamics outperform behavior cloning? *arXiv preprint arXiv:2601.21718*, 2026.

Gawon Seo, Dongwon Kim, and Suha Kwak. Acid: Action consistency via inverse dynamics for planning with world models, 2026. URL https://arxiv.org/abs/2607.02403.

Vlad Sobal, Wancong Zhang, Kyunghyun Cho, Randall Balestriero, Tim G. J. Rudner, and Yann LeCun. Stress-testing offline reward-free reinforcement learning: A case for planning with latent dynamics models. In *7th Robot Learning Workshop: Towards Robots with Human-Level Abilities*, 2025. URL https://openreview.net/forum?id=jON7H6A9UU.

Armin Sommer and Jannik Schilling. Reinforced planning with latent world models. *arXiv preprint arXiv:2608.18669*, 2026.

Hao Sun, Zhizhong Li, Xiaotong Liu, Bolei Zhou, and Dahua Lin. Policy continuation with hindsight inverse dynamics. *Advances in Neural Information Processing Systems*, 32, 2019.

Junhan Sun, Hao Zhao, and Guofeng Zhang. Intact: Isomorphic intent-to-action learning for search-free world models, 2026. URL https://arxiv.org/abs/2607.26056.

Basile Terver, Tsung-Yen Yang, Jean Ponce, Adrien Bardes, and Yann LeCun. What drives success in physical planning with joint-embedding predictive world models? *arXiv preprint arXiv:2512.24497*, 2025.

Yang Tian, Sizhe Yang, Jia Zeng, Ping Wang, Dahua Lin, Hao Dong, and Jiangmiao Pang. Predictive inverse dynamics models are scalable learners for robotic manipulation. In *International Conference on Learning Representations*, volume 2025, pp. 92033–92052, 2025.

Jiawei Wang, Ke Rui, Yushen Zuo, Yichun Feng, and Minglei Li. Decision-metric alignment in latent world models: Diagnostics and action-conditioned objectives for mpc planning. *arXiv preprint arXiv:2608.18746*, 2026a.

Yuhai Wang, Jiawei Xia, Rongxuan Zhou, Xiao Hu, Yongliang Shi, Jing Du, and Yang Ye. Prism: Prior-guided imagination sampling in world models. *arXiv preprint arXiv:2606.07974*, 2026b.

Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 1714–1721. IEEE, 2017.

Amber Xie, Oleh Rybkin, Dorsa Sadigh, and Chelsea Finn. Latent diffusion planning for imitation learning. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=vhACnRfuYh.

Zhenghao Zhang, Yuanxiang Wang, Zhenyu Guan, Yujia Yang, Bingkang Shi, Tianyu Zong, Hongzhu Yi, Guoqing Chao, Xingchen Chen, Tiankun Yang, Chenxi Bao, Tao Yu, Jingjing Zhou, and Jungang Xu. Delta-jepa: Learning action-sensitive world models via latent difference decoding, 2026. URL https://arxiv.org/abs/2606.31232.

Zhiwei Zhang, Hui Zhang, Kaihong Huang, Chenghao Shi, and Huimin Lu. Efficient image-goal navigation with representative latent world model, 2025. URL https://arxiv.org/abs/2511.11011.

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. URL https://openreview.net/forum?id=D5RNACOZEI.

APPENDIX

A DATASET DETAILS AND PROTOCOL

We evaluate on four continuous-control benchmarks: **Tworoom**, **Reacher**, **PushT**, and **OGBench-Cube** (hereafter OGCube), which together span 2D navigation, low-dimensional continuous control, planar object pushing, and robotic cube manipulation; Figure 3 shows a representative trajectory from each. We do not collect any trajectories ourselves: we reuse the full-scale offline datasets released by LeWM (Maes et al., 2026) without modification.

A.1 DATASETS

The four datasets are gathered under different behavior policies: a goal-directed navigation policy for Tworoom, a near-uniform random policy for Reacher, and expert policies for PushT and OGCube. This heterogeneity is directly visible in their action distributions (Figure 4): Reacher actions are essentially uniform over $[-1, 1]$, Tworoom actions saturate at the ± 1 boundaries, whereas PushT and OGCube produce concentrated, structured actions.

Each dataset is stored as a single HDF5 file under a shared schema: RGB observations `pixels` of shape $(N, 224, 224, 3)$ in uint8, a continuous `action` array of shape (N, A), and per-episode boundaries encoded by `ep_offset` and `ep_len`, so that a sampled window never straddles two episodes. Table 5 summarizes the resulting statistics.

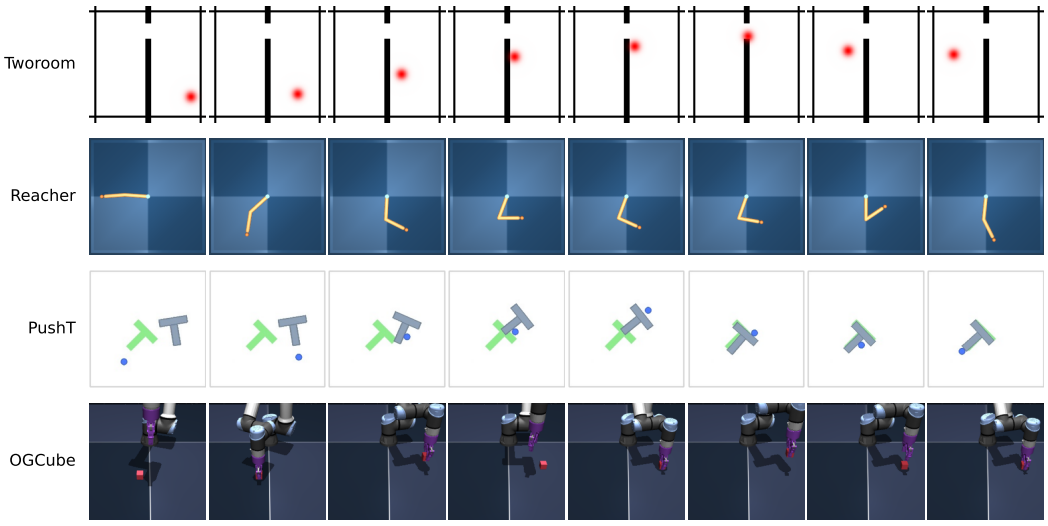

Figure 3: Task overview. Each row shows one representative trajectory per benchmark, sampled uniformly in time from its start (leftmost) to its end (rightmost).

Table 5: Statistics of the four datasets (full versions, as used in all experiments). Frames denotes the total number of transitions; Traj. length reports the minimum, mean, and maximum episode lengths.

Dataset	Simulator	Action dim A	#Episodes	Traj. length (min/mean/max)	#Frames
Tworoom	2D navigation	2	10,000	31 / 92 / 101	920,809
Reacher	DMControl Reacher	2	10,000	201 / 201 / 201	2,010,000
PushT	PushT	2	18,685	49 / 125 / 246	2,336,736
OGCube	OGBench Cube	5	10,000	201 / 201 / 201	2,010,000

Observations and actions. All observations are 224×224 RGB frames, which we feed to the encoder at full resolution. The action space is a 2D command for Tworoom, Reacher, and PushT,

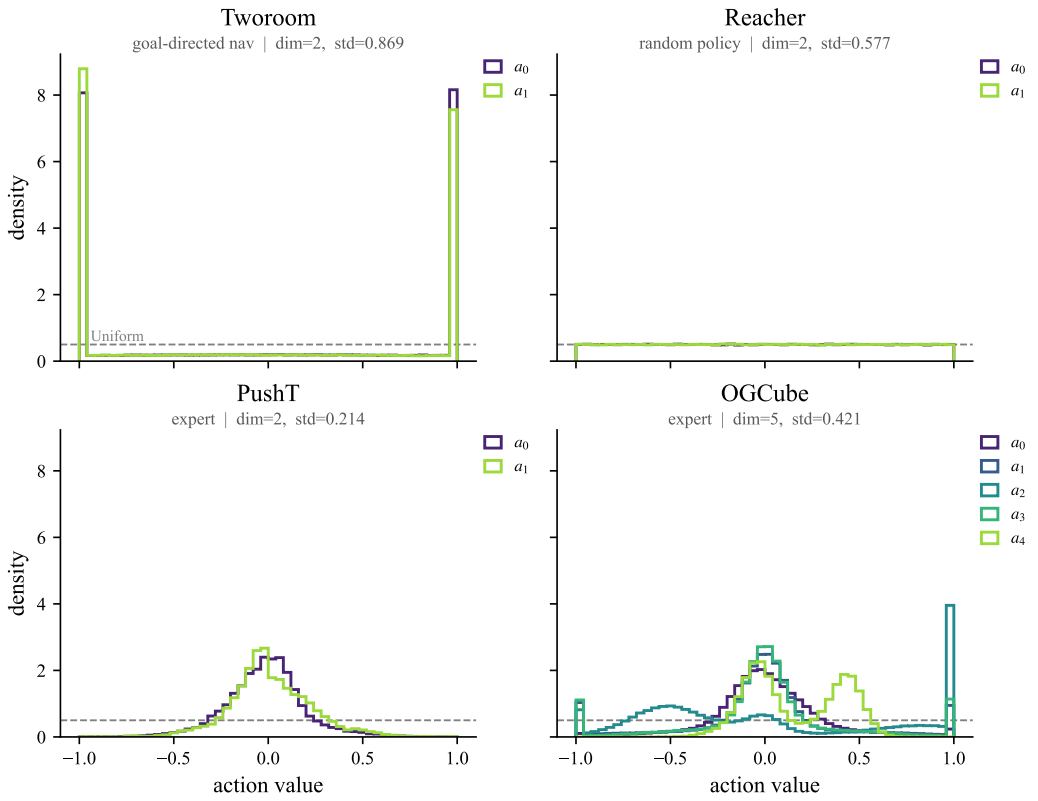

Figure 4: Per-dimension marginal action distributions for each dataset. The dashed line marks the density of a uniform distribution over $[-1, 1]$.

and a 5D vector (end-effector pose and gripper) for OGCube. Reacher and OGCube episodes have a fixed length of 201 steps, whereas Tworoom and PushT episodes vary in length because they terminate upon task completion.

Action normalization. Before training, every non-pixel input—the actions, and (where used) proprioception and state—is standardized per dimension by its dataset statistics, i.e., a z-score transform to zero mean and unit variance, applied identically to all four tasks. Figure 4 plots the corresponding *raw* (pre-normalization) action distributions, which differ markedly across tasks because of their different behavior policies.

Training samples and monitoring split. Following the data-loading and window-level splitting procedure of the LeWM implementation (Maes et al., 2026), we enumerate all valid windows within each trajectory and randomly assign 90% to training and 10% to validation. Each window contains W observations sampled at intervals of k raw environment steps, together with aligned action chunks. The loader requires Wk consecutive raw time-step records, while the first and last sampled observations are separated by $(W-1)k$ environment steps. We replace NaN action entries with zeros during training rather than dropping the window.

A.2 Evaluation Protocol

We follow the dataset-based evaluation procedure of the LeWM implementation (Maes et al., 2026). For each evaluation episode, we sample a start state from an offline trajectory and define the goal as the environment state K steps ahead (at index $\texttt{start} + K$). The agent acts for at most B steps. We count an episode as successful only if the simulator reports task completion within this budget Table 6; latent-space distance is not used to determine success.

Table 6: Environment-defined success criteria.

Task	Success condition
Tworoom	Agent-to-goal position distance < 16 in environment coordinates.
Reacher	Absolute error of every joint angle < 0.05 radians relative to the target configuration.
PushT	Euclidean error over agent and block positions < 20, and wrapped block orientation error $< \pi/9$.
OGCube	Target cube-to-goal position distance $\leq 0.04\,\mathrm{m}$.

Table 7: Difficulty tiers used in evaluation. Each tier is fully specified by the goal offset K (how many steps ahead of the start the goal is placed) and the step budget B (the maximum number of environment steps allowed).

Tier	Goal offset K	Step budget B
easy	10	25
default	25	50
hard	50	100
extreme	75	100

By varying the goal offset K and the step budget B, we obtain four difficulty tiers (Table 7; Figure 5), ranging from *easy* (K=10) to *extreme* (K=75): a larger K yields a longer-horizon goal. The extreme tier is the most challenging setting and is where our method attains its largest margin over the baselines.

A.3 CEM Planning Protocol

CEM plays several roles in our experiments. It is used as the test-time optimizer for the CEM-based world-model baselines, as the computational reference in the planning-efficiency experiment, and as a controlled planner ablation in which HIJP's amortized action generation is replaced by search over the same learned representation and forward dynamics. We therefore use one shared CEM configuration throughout all experiments.

At every replanning point, CEM maintains a factorized Gaussian distribution over action sequences of length $H = 5$ model steps. Each model-level action contains a chunk of $k = 5$ environment actions, so every candidate sequence covers $Hk = 25$ environment steps. The sampling distribution is initialized with zero mean and sampling scale 1.0.

At each optimization iteration, CEM samples $N = 300$ candidate action sequences and rolls each candidate forward through the corresponding latent dynamics model. Candidates are ranked by the squared distance between their terminal predicted latent and the encoded goal,

$$J_{\mathrm{goal}}(\mathbf{a}_{0:H-1}) = \|\hat{z}_H(\mathbf{a}_{0:H-1}) - z_g\|_2^2 . \tag{10}$$

The best 30 candidates are retained as elites and are used to refit the sampling mean and scale. This procedure is repeated for $I = 30$ iterations. Consequently, every CEM replanning call evaluates

$$NI = 300 \times 30 = 9{,}000 \tag{11}$$

candidate action sequences. Since every sequence contains $H = 5$ latent transitions, this corresponds to approximately

$$NIH = 300 \times 30 \times 5 = 45{,}000 \tag{12}$$

latent transition predictions per environment and replanning call. These operations are evaluated in parallel on the GPU; therefore, the sequence count should not be interpreted as an equivalent wall-clock slowdown.

All CEM comparisons use the same planning horizon, action chunking, interaction budget, and replanning cadence as HIJP. Both planners generate $H = 5$ model-level actions and execute $R = 5$ model steps, equivalent to $Rk = 25$ environment steps, before observing the environment and

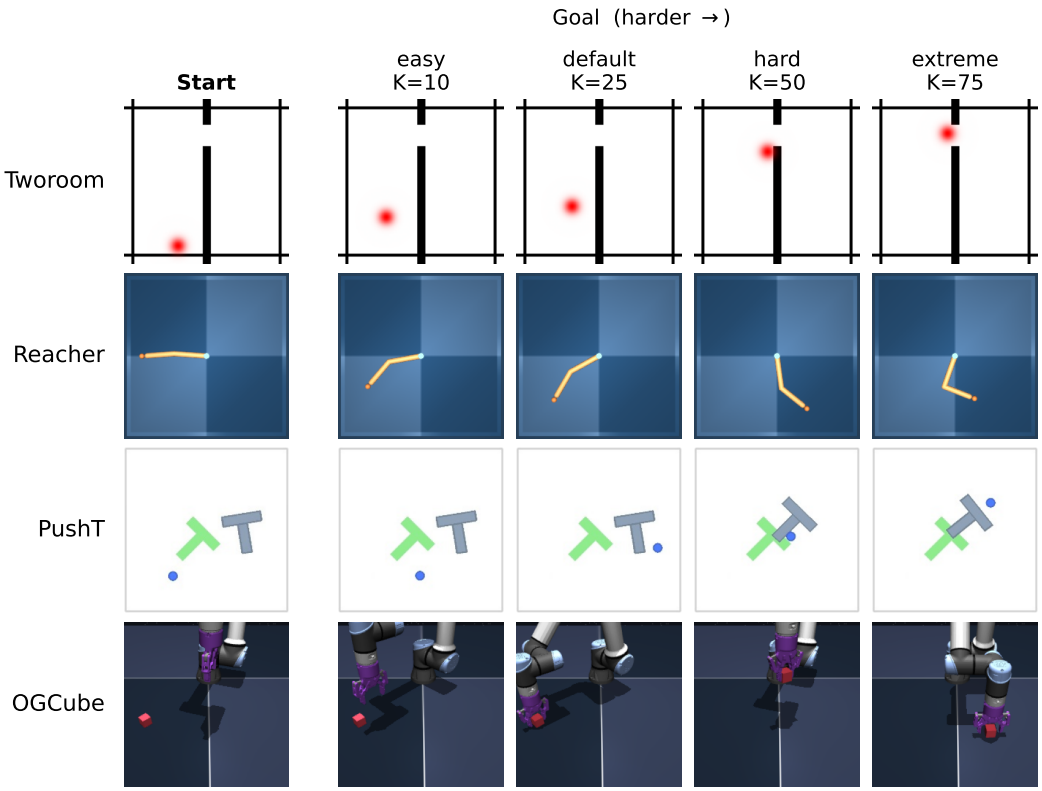

Figure 5: Visualization of the four difficulty tiers. For each task (row), we fix a start observation (leftmost column) and show the goal observation placed K steps ahead of it, one per tier. A larger offset K moves the goal farther along the trajectory and demands a longer planning horizon.

Table 8: CEM configuration shared across all CEM-based comparisons.

Hyperparameter	Value
Planning horizon H	5 model steps
Action-chunk length k	5 environment steps
Execution stride R	5 model steps
Environment steps per replan Rk	25
Candidate sequences N	300
Optimization iterations I	30
Elite sequences	30
Initial sampling scale	1.0
Sequence evaluations per replan	9,000
Latent transitions per replan	45,000

replanning. Under the interaction budgets $B \in \{25, 50, 100, 100\}$, the maximum numbers of replanning calls are therefore 1, 2, 4, and 4 for the easy, default, hard, and extreme tiers, respectively.

For the main LeWM baseline, CEM searches using LeWM's own encoder and forward model. In the controlled CEM ablation, it instead searches using HIJP's encoder and forward model while completely disabling the inverse-dynamics action generator. This controlled comparison changes only the test-time planning mechanism and therefore isolates the effect of amortized inverse-guided planning.

B Implementation Details

This appendix provides a concrete instantiation of the method. We first specify the temporal discretization and the architecture of each component, then describe the training procedure and test-time planning protocol. Unless stated otherwise, the architecture and temporal settings are shared across all four tasks.

B.1 Temporal Discretization and Data Flow

The environments expose atomic actions $u_s \in \mathbb{R}^{d_a}$ at raw environment step s. The world model operates at a coarser temporal resolution by grouping k consecutive atomic actions into one model-level action chunk, $\mathbf{a}_t = (u_{tk}, u_{tk+1}, \ldots, u_{tk+k-1}) \in \mathbb{R}^{kd_a}$.

We use $k = 5$ for every task. Consequently, one transition in the learned latent dynamics corresponds to five raw environment steps. The raw action dimension is $d_a = 2$ for Tworoom, Reacher, and PushT, and $d_a = 5$ for OGCube. The effective model-level action dimension is therefore $d_{\mathbf{a}} = kd_a \in \{10, 25\}$.

A training sample contains W temporally subsampled observations, together with their aligned model-level action chunks. $k = 5$ raw environment steps separate consecutive observations. Thus, the first and final observations in a window are separated by $(W - 1)k$ raw environment steps.

Before being passed to the model, images are converted to floating-point, normalized using ImageNet statistics, and resized to 224×224. Following the LeWM preprocessing procedure, continuous actions are standardized per dimension using statistics computed from the training dataset. At execution time, predicted action chunks are mapped back to the environment action space using the inverse transformation.

B.2 Architecture

The implementation contains a visual encoder f_θ, an action encoder ψ_ω, a latent predictor g_ϕ, and a hindsight-conditioned inverse-dynamics head h_ξ. The action encoder is absorbed into the notation of g_ϕ in the main text and is made explicit here.

Visual encoder f_θ. We instantiate f_θ as a ViT-Tiny(Dosovitskiy et al., 2021) trained from scratch, without a pretrained backbone or an external visual representation. It contains 12 Transformer layers, 3 attention heads, and a hidden width of 192. Each 224×224 observation is divided into patches of size 14×14. We take the final `[CLS]` token and pass it through a projection MLP $192 \to 2048 \to 192$. Batch normalization is applied at the hidden layer of the projection head. The resulting vector is the latent representation $z_t = f_\theta(o_t) \in \mathbb{R}^{192}$.

Action encoder ψ_ω. The flattened action chunk $\mathbf{a}_t \in \mathbb{R}^{d_{\mathbf{a}}}$ is embedded in two stages. A pointwise linear projection, implemented as a 1×1 one-dimensional convolution, first maps $d_{\mathbf{a}} \to 10$. A two-layer MLP then maps $10 \to 768 \to 192$, with a SiLU activation between the two linear layers. This produces an action embedding with the same width as the visual latent, $e_t^a = \psi_\omega(\mathbf{a}_t) \in \mathbb{R}^{192}$.

latent predictor g_ϕ. The predictor is a six-layer autoregressive Transformer with causal self-attention. It operates at width 192, uses 16 attention heads with head dimension 64, an MLP width of 2048, attention and MLP dropout of 0.1, and learned positional embeddings. The predictor receives a sequence of latent tokens and a time-aligned sequence of action embeddings. Actions condition every Transformer block through AdaLN-Zero(Peebles & Xie, 2023). In particular, each action embedding produces the shift, scale, and residual-gating parameters for both the attention and feed-forward sublayers; the modulation layer is initialized to zero.

The predictor uses at most the $m = 3$ most recent latent-action pairs. For a context ending at time t, its last output is interpreted as the prediction of the next latent,$\hat{z}_{t+1} = g_\phi(z_{t-m+1:t}, \psi_\omega(\mathbf{a}_{t-m+1:t}))$,where the context is shortened near the beginning of a sequence. The Transformer output is mapped to the latent space by a second projection MLP $192 \to 2048 \to 192$ with batch normalization.

Table 9: Architecture used in all experiments.

Component	Configuration
Latent width	$d = 192$
Visual encoder f_θ	ViT-Tiny; 12 layers; 3 heads; width 192; patch size 14; 224×224 inputs; trained from scratch
Visual projection	MLP $192 \to 2048 \to 192$ with batch normalization
Action encoder ψ_ω	1×1 convolution $d_{\mathbf{a}} \to 10$, followed by MLP $10 \to 768 \to 192$ with SiLU
latent predictor g_ϕ	Causal Transformer; 6 layers; 16 heads; head dimension 64; MLP width 2048; dropout 0.1; AdaLN-Zero conditioning
Predictor context	$m = 3$ latent–action pairs
Predictor projection	MLP $192 \to 2048 \to 192$ with batch normalization
Inverse head h_ξ	Three hidden layers of width 512, each with LayerNorm and GELU
Frameskip	$k = 5$ raw environment steps per model step

Inverse-dynamics head h_ξ. The inverse head receives the concatenation of a current latent and a future goal latent,

$$h_\xi : \mathbb{R}^{2d} \to \mathbb{R}^{d_{\mathbf{a}}}, \qquad \hat{\mathbf{a}}_i = h_\xi(z_i, z_{i+\kappa}). \tag{13}$$

It consists of three hidden layers of width 512. Each hidden layer applies a linear transformation, LayerNorm, and GELU activation. A final linear layer maps to the model-level action dimension $d_{\mathbf{a}}$.

B.3 Training

All four modules are optimized jointly. Training combines next-latent prediction, anti-collapse representation regularization, hindsight-conditioned inverse dynamics, and action transition consistency.

latent prediction loss. Although every sampled window contains $W = 16$ model steps, the prediction term uses a causal context of length $m = 3$. Specifically, the predictor consumes $z_{0:m-1}$ and the corresponding action embeddings and predicts the shifted latent sequence $z_{1:m}$. Because causal attention is used, the output at position t only depends on positions up to t. Denoting the three predicted latents by $\hat{z}_{1:m}$, we compute

$$\mathcal{L}_{\text{pred}} = \frac{1}{Bmd} \sum_{b=1}^{B} \sum_{t=0}^{m-1} \left\| \hat{z}_{t+1}^{(b)} - z_{t+1}^{(b)} \right\|_2^2. \tag{14}$$

The target latent in this term remains attached to the computational graph, for which we do not employ a momentum target encoder.

Anti-collapse regularization. We apply the Sketch Isotropic Gaussian Regularizer (SIGReg; Balestriero & LeCun, 2025) to the latents from all W frames, including frames that are not used by Equation 14. SIGReg measures the discrepancy between random one-dimensional projections of the latent distribution and a standard Gaussian reference distribution. We use 17 integration knots and 1024 random projections. Applying this objective over the complete window increases the number and temporal diversity of latent samples contributing to the regularizer.

Hindsight-conditioned inverse dynamics. For each training window, we independently sample N_a anchor–goal pairs. For the n-th pair, the anchor is sampled uniformly from all positions with an available successor,

$$i_n \sim \text{Unif}\{0, \ldots, W-2\}. \tag{15}$$

Conditioned on i_n, we sample a future offset uniformly from the remaining part of the window,

$$\kappa_n \sim \text{Unif}\{1, \ldots, W-1-i_n\}. \tag{16}$$

The resulting goal index is $i_n + \kappa_n$. Sampling is performed independently for each pair and each element of the minibatch, so anchors may repeat. This procedure exposes the inverse head to goal conditions at multiple temporal distances while requiring no rewards or task labels.

Table 10: Task-specific training hyperparameters for HIJP.

Task	Batch	Learning rate	N_a	λ_{reg}	λ_{inv}	λ_{cyc}	Epochs	Scheduler
TwoRoom	48	3×10^{-5}	4	0.09	1.0	0.05	10	LWC
Reacher	48	5×10^{-5}	4	0.09	1.0	0.10	10	LWC
PushT	48	5×10^{-5}	4	0.09	1.0	0.10	10	LWC
OGCube	32	5×10^{-5}	4	0.09	1.0	0.10	10	LWC

For every sampled pair, the inverse head predicts the demonstrated action chunk executed at the anchor,

$$\hat{\mathbf{a}}_{i_n} = h_\xi(z_{i_n}, z_{i_n+\kappa_n}). \tag{17}$$

Importantly, the target is the immediate action chunk $\mathbf{a}_{i_n}$, not the sequence of actions between the anchor and the sampled goal. The inverse loss is the element-wise mean absolute error

$$\mathcal{L}_{\text{inv}} = \frac{1}{BN_a d_{\mathbf{a}}} \sum_{b=1}^{B} \sum_{n=1}^{N_a} \left\| \hat{\mathbf{a}}_{i_n}^{(b)} - \mathbf{a}_{i_n}^{(b)} \right\|_1 . \tag{18}$$

Action transition consistency loss. The action consistency term couples the inverse head to the learned latent prediction. For each hindsight pair, we embed the predicted action and perform a single predictor step from the anchor latent,

$$\tilde{z}_{i_n+1} = g_\phi(z_{i_n}, \psi_\omega(\hat{\mathbf{a}}_{i_n})) . \tag{19}$$

We then regress the prediction toward the actual next latent using

$$\mathcal{L}_{\text{cyc}} = \frac{1}{BN_a d} \sum_{b=1}^{B} \sum_{n=1}^{N_a} \left\| \tilde{z}_{i_n+1}^{(b)} - \text{sg}\left[z_{i_n+1}^{(b)} \right] \right\|_2^2 , \tag{20}$$

where sg denotes stop-gradient. Gradients from this loss update h_ξ, ψ_ω, g_ϕ, and f_θ, as well as the anchor-side representation through z_{i_n}, while the next-latent target is detached. The single-step context used by this term is supported by the same autoregressive predictor used for the length-m forward objective.

Joint objective. The complete objective is

$$\mathcal{L} = \mathcal{L}_{\text{pred}} + \lambda_{\text{reg}}\mathcal{L}_{\text{sigreg}} + \lambda_{\text{inv}}\mathcal{L}_{\text{inv}} + \lambda_{\text{cyc}}\mathcal{L}_{\text{cyc}}. \tag{21}$$

All terms are averaged over the batch and their corresponding temporal, anchor, action, or latent dimensions before the weights are applied.

Optimization. We use a single AdamW optimizer for f_θ, ψ_ω, g_ϕ, and h_ξ, with weight decay 10^{-3}. Training uses bf16 mixed precision and a global gradient-norm clipping threshold of 1.0. The learning rate follows a linear-warmup cosine-decay schedule, starting from zero. With ten training epochs, the configured warmup is rounded to one epoch, followed by cosine decay to zero. Training and validation use the LeWM-based window-level split described in the dataset appendix A, with a 90%/10% ratio and random seed 3072.

Algorithm 1 summarizes the complete training procedure, and Table 10 reports the task-specific training parameters.

B.4 Planning and Receding-Horizon Execution

At each replanning point, the current observation and target observation are encoded as

$$z_0 = f_\theta(o_t), \qquad z_g = f_\theta(o_g). \tag{22}$$

Starting from z_0, we construct an H-step action sequence through the autoregressive latent rollout introduced in the main text. At rollout step j, the inverse head first predicts the action chunk that is consistent with moving from the current imagined latent toward the goal latent,

$$\mathbf{a}_j = h_\xi(z_j, z_g). \tag{23}$$

Algorithm 1 Joint training with hindsight-conditioned inverse dynamics

Require: Offline dataset $\mathcal{D}$; window length W; predictor context length m; number of hindsight pairs N_a

for each training minibatch **do**
  Sample a minibatch of windows $(o_{0:W-1}, \mathbf{a}_{0:W-2}) \sim \mathcal{D}$
  Encode observations and actions: $z_t \leftarrow f_\theta(o_t)$ and $e_t^a \leftarrow \psi_\omega(\mathbf{a}_t)$
  Predict the next latents: $\hat{z}_{1:m} \leftarrow g_\phi(z_{0:m-1}, e^a_{0:m-1})$
  $\mathcal{L}_{\text{pred}} \leftarrow \text{MSE}(\hat{z}_{1:m}, z_{1:m})$
  $\mathcal{L}_{\text{sigreg}} \leftarrow \text{SIGReg}(z_{0:W-1})$
  for $n = 1, \dots, N_a$ **do**
    Sample $i_n \sim \text{Unif}\{0, \dots, W-2\}$
    Sample $\kappa_n \sim \text{Unif}\{1, \dots, W-1-i_n\}$
    Predict the demonstrated action: $\hat{\mathbf{a}}_{i_n} \leftarrow h_\xi(z_{i_n}, z_{i_n+\kappa_n})$
    Predict the successor latent: $\tilde{z}_{i_n+1} \leftarrow g_\phi(z_{i_n}, \psi_\omega(\hat{\mathbf{a}}_{i_n}))$
  end for
  $\mathcal{L}_{\text{inv}} \leftarrow \text{MAE}\Big(\{\hat{\mathbf{a}}_{i_n}\}_{n=1}^{N_a}, \{\mathbf{a}_{i_n}\}_{n=1}^{N_a}\Big)$
  $\mathcal{L}_{\text{cyc}} \leftarrow \text{MSE}\Big(\{\tilde{z}_{i_n+1}\}_{n=1}^{N_a}, \{\text{sg}[z_{i_n+1}]\}_{n=1}^{N_a}\Big)$
  $\mathcal{L} \leftarrow \mathcal{L}_{\text{pred}} + \lambda_{\text{reg}}\mathcal{L}_{\text{sigreg}} + \lambda_{\text{inv}}\mathcal{L}_{\text{inv}} + \lambda_{\text{cyc}}\mathcal{L}_{\text{cyc}}$
  Update $(\theta, \omega, \phi, \xi)$ with AdamW
end for

The action chunk is embedded and passed to the latent predictor, which advances the imagined state,

$$z_{j+1} = g_\phi\big(z_{\max(0,j-m+1):j}, \psi_\omega(\mathbf{a}_{\max(0,j-m+1):j})\big)\,. \tag{24}$$

The available context grows from one to m latent-action pairs during the first rollout steps and is subsequently truncated to the most recent $m = 3$ pairs. Thus, constructing an H-step plan requires H evaluations of the inverse head and H autoregressive evaluations of the latent predictor. No population of candidate action sequences or iterative test-time action optimization is required.

We use $H = 5$ model steps for every task. Since each model-level action contains $k = 5$ atomic actions, one predicted plan covers $Hk = 25$ raw environment steps. The controller operates in a receding-horizon loop with execution stride $R = 5$ model steps. It therefore executes the complete predicted action sequence, re-encodes the resulting observation, and constructs a new plan. The target observation remains unchanged within an episode but is encoded together with the current observation at each replanning point. Execution terminates when the task-specific success condition is satisfied, or the allotted budget of raw environment steps is exhausted. Algorithm 2 gives the complete procedure.

Table 11 collects the planning settings shared across the four tasks. Keeping these settings unchanged ensures that differences across tasks do not arise from task-specific planning budgets or model-rollout depths.

Table 11: Planning settings shared across all four tasks.

Hyperparameter	Value
Planning horizon H	5 model steps
Action-chunk length k	5 raw steps
Raw planning horizon Hk	25 environment steps
Execution stride R	5 model steps
Predictor context m	3 latent–action pairs

C Loss Weight Ablation

To examine the sensitivity of HIJP to the weighting coefficients of its training objectives, we conduct ablations on the inverse dynamics loss weight λ_{inv} and the cycle-consistency loss weight λ_{cyc}.

Algorithm 2 Amortized latent planning

Require: Trained f_θ, ψ_ω, g_ϕ, and h_ξ; goal observation o_g; planning horizon H; execution stride R; action-chunk length k; predictor context length m; raw-step budget B

$n \leftarrow 0$
while $n < B$ and the task has not succeeded **do**
  Encode the current and goal observations: $\hat{z}_0 \leftarrow f_\theta(o_t)$, $z_g \leftarrow f_\theta(o_g)$
  Initialize $\mathcal{Z} \leftarrow [\hat{z}_0]$ and $\mathcal{A} \leftarrow [\,]$
  for $j = 0, \ldots, H-1$ **do**
    Predict an action chunk: $\mathbf{a}_j \leftarrow h_\xi(\hat{z}_j, z_g)$
    Append $\mathbf{a}_j$ to $\mathcal{A}$
    Predict the next imagined latent:

$$\hat{z}_{j+1} \leftarrow g_\phi(\mathrm{tail}_m(\mathcal{Z}), \psi_\omega(\mathrm{tail}_m(\mathcal{A})))$$

    Append $\hat{z}_{j+1}$ to $\mathcal{Z}$
  end for
  $q \leftarrow \min\left\{R,\ H,\ \left\lfloor \frac{B-n}{k} \right\rfloor\right\}$
  Execute the first q action chunks in $\mathcal{A}$, terminating early if the task succeeds
  Let s be the number of raw environment steps actually executed
  $n \leftarrow n + s$
end while

Table 12: Success rates (%) for the loss-weight ablations. Avg. reports the arithmetic mean across the four difficulty tiers, and Overall Avg. averages all 12 task–tier combinations. Bold column headers indicate the default weights. In the Overall Avg. row, the best and second-best results within each weight group are shown in **bold** and underlined text.

		λ_{inv} ($\lambda_{\text{cyc}} = 0.1$)				λ_{cyc} ($\lambda_{\text{inv}} = 1.0$)			
Task	Tier	0	0.1	**1.0**	2.0	0	0.01	**0.1**	0.2
Reacher	Easy	36%	42%	48%	44%	34%	52%	48%	44%
	Default	72%	88%	92%	78%	78%	86%	92%	64%
	Hard	90%	86%	92%	90%	84%	90%	92%	78%
	Extreme	86%	98%	94%	90%	94%	90%	94%	92%
	Avg.	71.0%	78.5%	81.5%	75.5%	72.5%	79.5%	81.5%	69.5%
PushT	Easy	36%	90%	90%	92%	84%	90%	90%	92%
	Default	6%	78%	80%	82%	78%	72%	80%	82%
	Hard	0%	56%	66%	62%	54%	56%	66%	68%
	Extreme	0%	36%	42%	58%	22%	58%	42%	44%
	Avg.	10.5%	65.0%	69.5%	73.5%	59.5%	69.0%	69.5%	71.5%
OGCube	Easy	80%	94%	98%	100%	98%	96%	98%	92%
	Default	56%	94%	96%	94%	96%	94%	96%	92%
	Hard	46%	90%	96%	96%	94%	96%	96%	94%
	Extreme	40%	96%	96%	96%	96%	94%	96%	94%
	Avg.	55.5%	93.5%	96.5%	96.5%	96.0%	95.0%	96.5%	93.0%
Overall Avg.		45.7%	79.0%	**82.5%**	81.8%	76.0%	81.2%	**82.5%**	78.0%

Specifically, we vary $\lambda_{\text{inv}} \in \{0, 0.1, 1.0, 2.0\}$ while fixing $\lambda_{\text{cyc}} = 0.1$, and vary $\lambda_{\text{cyc}} \in \{0, 0.01, 0.1, 0.2\}$ while fixing $\lambda_{\text{inv}} = 1.0$. We train a separate model for each weight setting and evaluate the resulting checkpoint using the same evaluation protocol as in the main experiments. All training and evaluation settings follow Section 4.1. We report the results on Reacher, PushT, and OGCube in Table 12. We omit TwoRoom because its performance is already saturated under our evaluation protocol, with near 100% success rates, making it uninformative for distinguishing different loss-weight choices.

The default setting, $\lambda_{\text{inv}} = 1.0$ and $\lambda_{\text{cyc}} = 0.1$, achieves the highest overall average success rate of 82.5%. Increasing λ_{inv} to 2.0 improves performance on PushT and matches the default performance on OGCube, but reduces performance on Reacher, resulting in a slightly lower overall average of 81.8%. Similarly, decreasing λ_{cyc} to 0.01 or increasing it to 0.2 yields overall averages of 81.2% and 78.0%, respectively. These results indicate that the default weights provide the best overall balance across the three evaluated environments.

D ACTION SMOOTHNESS EXPERIMENT

This appendix provides the complete evaluation protocol, metric definition, and per-tier results for the action-smoothness study in Section 4.6. We evaluate HIJP, its CEM ablation, and LeWM on the four domains described in Appendix A.

For each domain and difficulty tier, all methods are evaluated closed-loop on the same set of start–goal instances. Instance sampling depends only on the dataset, random seed, goal offset, and number of evaluation episodes, and is therefore independent of the evaluated method. We use 50 episodes per task with random seed 42. The four difficulty tiers are defined by the goal-offset and step-budget pairs (K, B) listed in Table 7. Each rollout is allowed to run for the corresponding step budget, and we record the action executed at every environment step.

HIJP directly executes the amortized plan produced by its inverse-guided controller and performs no test-time search. The CEM ablation uses the same learned representation and predictive model as HIJP, but replaces inverse-guided action generation with CEM planning. LeWM, which performs CEM planning over its own forward model, is included as an additional reference. All methods use the same control frequency and normalized action space.

Table 13: Action jerk across four environments and four difficulty tiers. Avg. reports the arithmetic mean across tiers, with average ratios computed from the per-tier ratios. The lowest jerk in each row is shown in **bold**.

		HIJP	CEM (ablation)		LeWM	
Environment	Tier	Jerk ↓	Jerk ↓	Ratio ↑	Jerk ↓	Ratio ↑
PushT	easy	**0.038**	0.439	11.5×	0.451	11.8×
	default	**0.040**	0.450	11.2×	0.407	10.1×
	hard	**0.048**	0.611	12.8×	0.595	12.4×
	extreme	**0.048**	0.697	14.7×	0.724	15.2×
	Avg.	**0.043**	0.549	12.6×	0.544	12.4×
TwoRoom	easy	**0.082**	1.838	22.4×	1.774	21.6×
	default	**0.084**	2.216	26.3×	2.191	26.0×
	hard	**0.120**	2.325	19.4×	2.428	20.2×
	extreme	**0.146**	2.503	17.1×	2.558	17.5×
	Avg.	**0.108**	2.220	21.3×	2.238	21.3×
Reacher	easy	**0.040**	1.179	29.5×	1.099	27.5×
	default	**0.046**	1.174	25.6×	1.114	24.3×
	hard	**0.046**	1.200	26.3×	1.134	24.8×
	extreme	**0.046**	1.222	26.4×	1.134	24.5×
	Avg.	**0.045**	1.194	27.0×	1.120	25.3×
OGCube	easy	**0.126**	1.596	12.7×	1.517	12.1×
	default	**0.073**	1.843	25.2×	1.886	25.8×
	hard	**0.088**	1.848	21.0×	2.159	24.5×
	extreme	**0.099**	2.065	20.9×	2.304	23.3×
	Avg.	**0.097**	1.838	19.9×	1.967	21.4×

If an episode reaches its goal before exhausting the step budget, we truncate its action sequence at the first success step and compute smoothness only over the executed prefix. This avoids either rewarding or penalizing post-success idling and measures the controls actually required to complete the task. Unsuccessful episodes contribute their full budget-length action sequences.

Let $a_{1:T} = (a_1, \ldots, a_T)$ denote the executed action sequence of an episode. We define *action jerk* as the mean ℓ_2 norm of the discrete second difference:

$$\mathrm{Jerk}(a_{1:T}) = \frac{1}{T-2} \sum_{t=3}^{T} \|a_t - 2a_{t-1} + a_{t-2}\|_2 \,. \tag{25}$$

Lower values indicate smoother controls. A temporally constant or linearly varying action sequence has zero second difference, whereas oscillatory or abruptly changing commands produce a large value.

For each method, environment, and difficulty tier, we compute Eq. 25 independently for every episode and report the mean across episodes. Let $\overline{J}_m$ denote this episode-averaged jerk for method m. We additionally report

$$R_{\mathrm{CEM}} = \frac{\overline{J}_{\mathrm{CEM}}}{\overline{J}_{\mathrm{HIJP}}}, \qquad R_{\mathrm{LeWM}} = \frac{\overline{J}_{\mathrm{LeWM}}}{\overline{J}_{\mathrm{HIJP}}}. \tag{26}$$

A ratio greater than one indicates that HIJP produces smoother actions than the corresponding CEM-based planner. Because all methods are evaluated in the same normalized action space, their jerk values are directly comparable.

Table 13 reports the results across all four difficulty tiers. HIJP consistently achieves lower action jerk than both its CEM ablation and LeWM. On the extreme tier ($K = 75$), the CEM ablation produces $14.7\times$–$26.4\times$ higher jerk than HIJP, despite sharing the same learned representation and predictive model. This controlled comparison shows that the smoothness advantage primarily arises from HIJP's amortized inverse-guided planning rather than from a difference in learned representations. LeWM exhibits a similar level of jerk to the CEM ablation, suggesting that independently optimizing successive plans with CEM can introduce substantial temporal discontinuities into the executed controls.

E Visualizing Planned Trajectories

For each environment, we run our planner (Algorithm 2) from a start observation toward a goal observation and record the resulting trajectory. Figures 6–9 visualize these rollouts as pixel-space filmstrips.

Each figure corresponds to one environment and contains four difficulty tiers: Easy, Default, Hard, and Extreme. The tiers are defined by goal offsets of 10, 25, 50, and 75 environment steps, with corresponding planning budgets of 25, 50, 100, and 100 steps, respectively. For each tier, we show one rollout using 16 frames sampled uniformly over its execution, from 0% to 100%, and arrange them in two rows of eight frames. The goal observation appears separately on the right, outlined in red. The label beside each tier indicates whether the rollout reaches the goal within the planning budget.

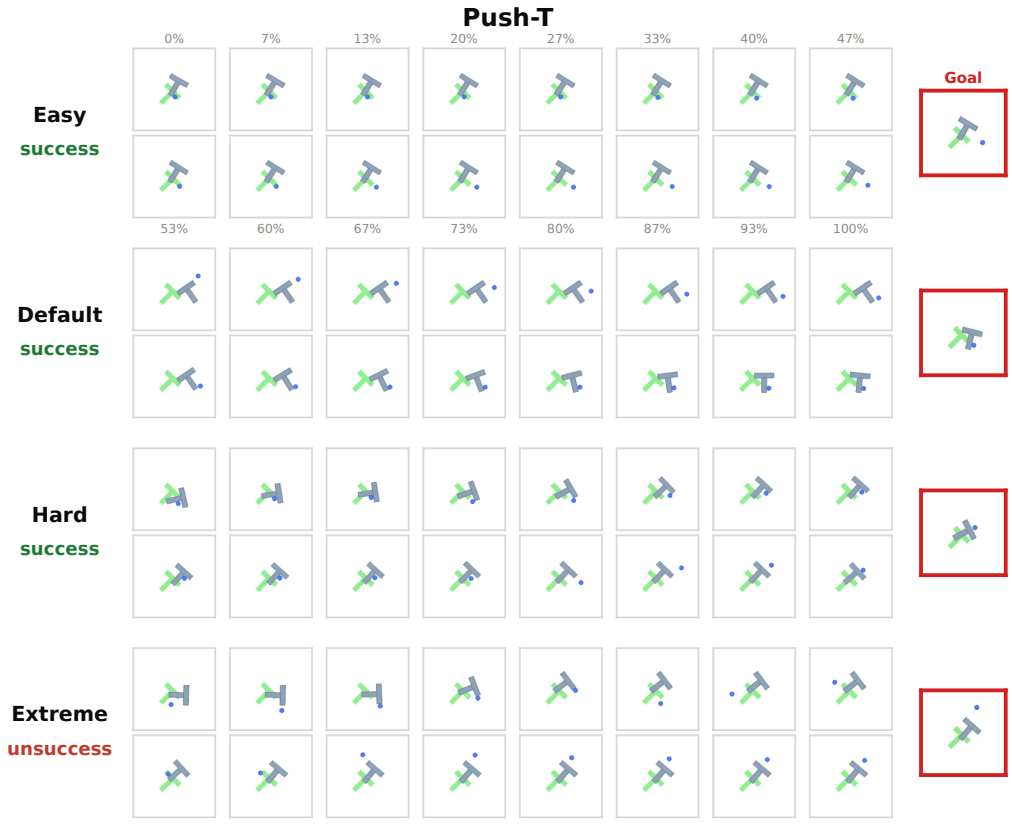

Figure 6: Representative trajectories produced by HIJP in the Push-T environment.

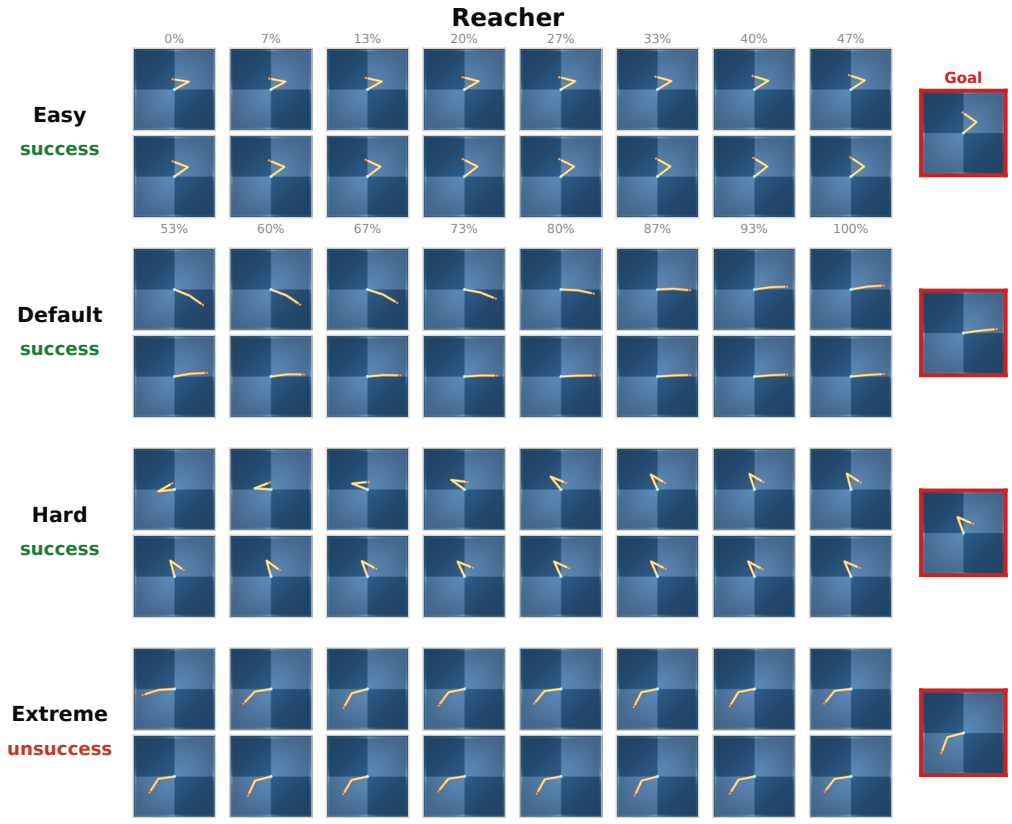

Figure 7: Representative trajectories produced by HIJP in the Reacher environment.

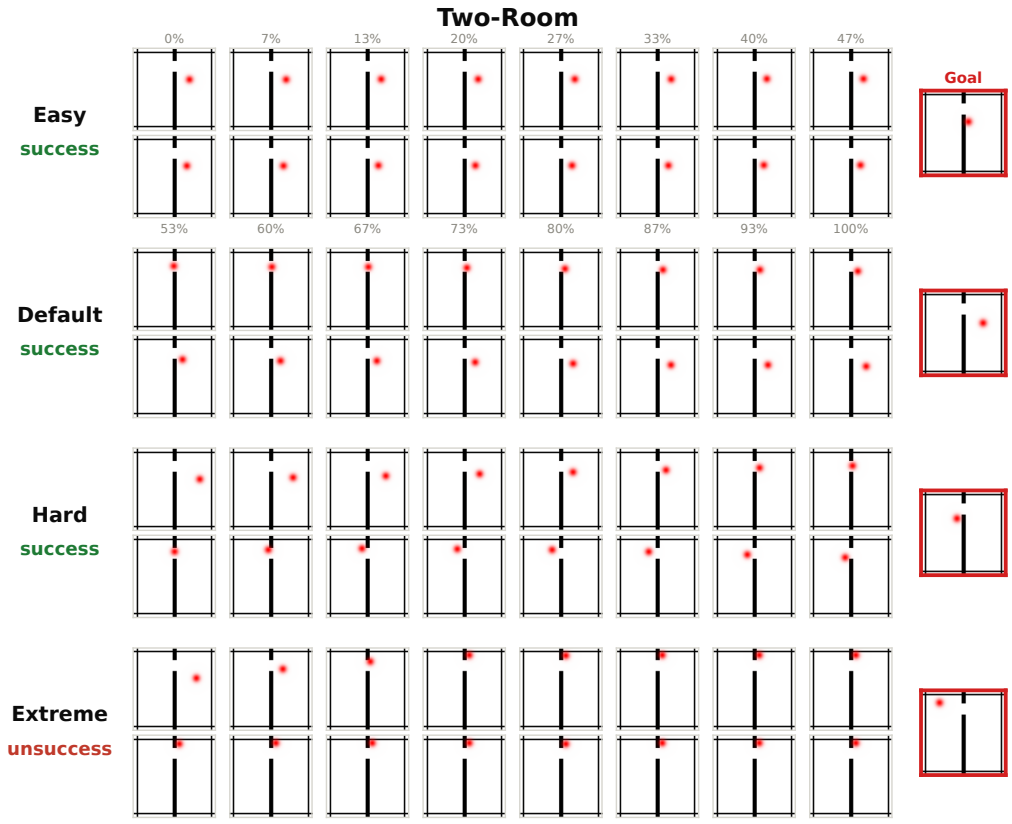

Figure 8: Representative trajectories produced by HIJP in the TwoRoom environment.

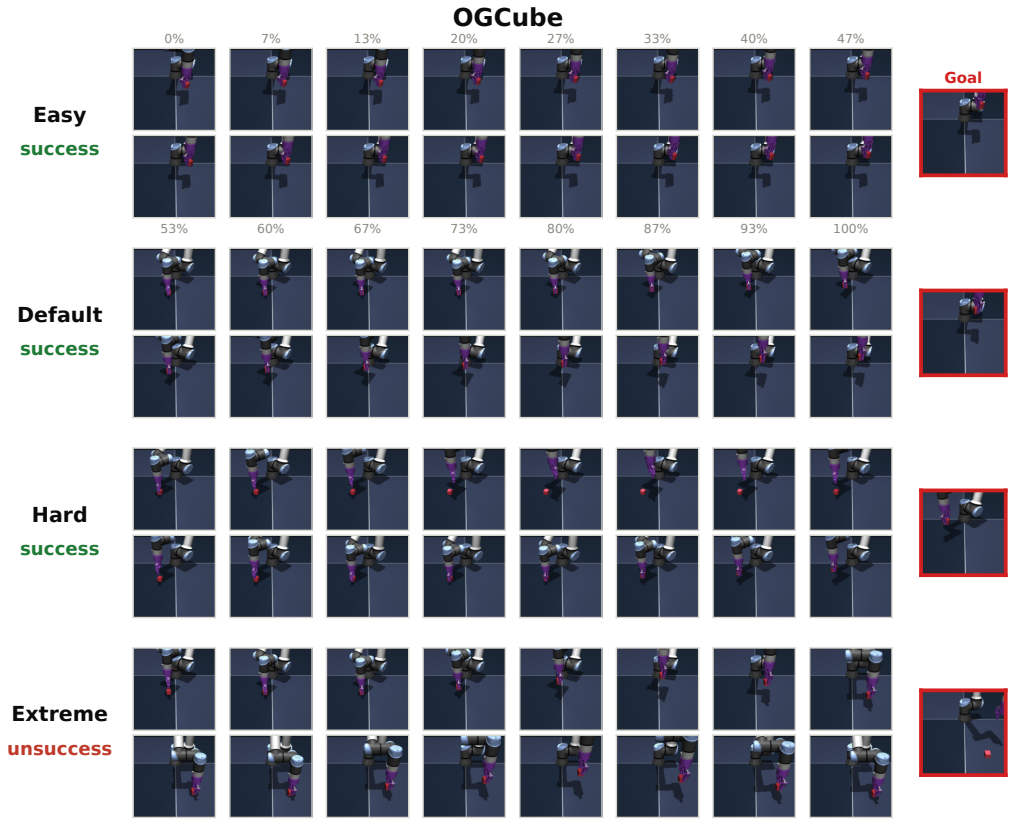

Figure 9: Representative trajectories produced by HIJP in the OGCube environment.