PARALLEL PREDICTIVE WORLD MODELS FOR ACCURATE AND EFFICIENT LONG-HORIZON PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon world-model planning typically relies on autoregressive rollouts, where predicted states are repeatedly fed back into the model. This preserves temporal structure but creates a horizon-length sequential path and exposes later predictions to recursive decoded-state feedback. We introduce Parallel Predictive World Models (PPWM), which predict a finite-horizon trajectory in parallel while retaining causal interaction among future representations. Each horizon is conditioned on its causal action prefix, and future representations interact before decoding, separating temporal causality from state-by-state output recursion. We formalize this distinction by viewing autoregressive rollout as a causal trajectory map and identifying the decoded-state feedback pathway removed by PPWM. Across four visual-control tasks, PPWM achieves the lowest long-horizon prediction error and the highest Cross-Entropy Method (CEM) simulator success among the evaluated predictive interfaces. Meanwhile, PPWM achieves more than a $3\times$ average CEM planning speedup over the autoregressive LeWM baseline. These results suggest that accurate and efficient long-horizon world-model planning does not require state-by-state autoregression, but can instead be achieved through parallel causal trajectory prediction.

1 INTRODUCTION

World models let agents evaluate candidate behaviors through imagined futures before acting. Effective planning often requires sufficiently long futures, because the consequences of early actions may emerge only many steps later. Learned dynamics have therefore been paired with trajectory optimization and model-predictive control (Chua et al., 2018; Hansen et al., 2022; 2024), while recent sequence and visual world models extend this paradigm to high-dimensional observations and offline planning (Micheli et al., 2022; Zhou et al., 2024; Sobal et al., 2026). As horizons grow, world-model planning must preserve predictive accuracy while avoiding the increasing sequential cost of state-by-state rollout.

Most world models expose future dynamics through recursive local prediction. Distant states require a horizon-dependent chain of model evaluations, and each predicted state becomes part of the next model input, allowing prediction error or model misspecification to propagate through the rollout (Hafner et al., 2019; Janner et al., 2019; Gao and Xu, 2026; Somalwar et al., 2025). Direct multi-step prediction provides an alternative factorization that can mitigate recursive bias under partial observability or model misspecification (Somalwar et al., 2025; 2026). Crucially, downstream planners inherit the same temporal factorization when evaluating long trajectories or propagating the influence of early actions to distant outcomes.

Recent work reduces this dependency from two sides. Prediction-side methods directly parameterize longer futures (Mishra et al., 2017). Most closely, Fast-LeWM accumulates action effects with causal action prefixes and predicts multiple horizons in parallel (Gao and Xu, 2026), but does not explicitly couple future-state representations before decoding. Planning-side methods instead lift intermediate states to expose trajectory-level parallelism (Rybkin et al., 2021; Ziakas et al., 2026; Wu et al., 2026). GRASP performs parallel iterative optimization with virtual states while avoiding long chains of state-input Jacobians through an action-gradient formulation (Psenka et al., 2026). This leaves a concrete design problem: preserve trajectory causality without forcing prediction and planning through a state-by-state temporal chain.

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

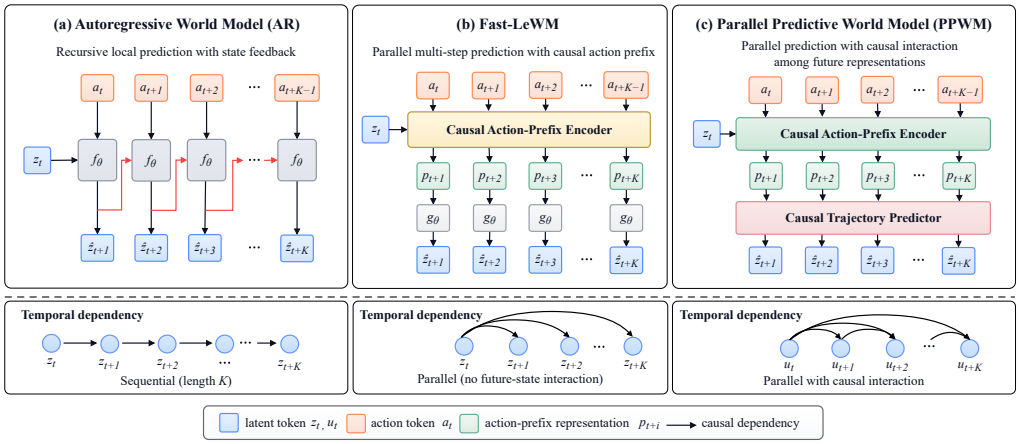

Figure 1: Comparison of temporal interfaces for world models. (a) Autoregressive (AR) world models predict one step at a time and feed the predicted state back as input, forming a long temporal chain. (b) Fast-LeWM uses a causal action prefix to predict multiple future states in parallel, without explicit interaction among future-state representations. (c) Our Parallel Predictive World Model (PPWM) predicts a causal finite-horizon trajectory in parallel, allowing interaction among future representations while removing output-state recursion within a span.

We introduce Parallel Predictive World Models (PPWM) for long-horizon planning. Given recent context and an action span, PPWM predicts a causally structured finite-horizon trajectory directly, without output-state recursion inside the span and with causal interaction among future representations. This design induces a parallel trajectory interface that can be queried by population-based search and differentiated jointly for lightweight trajectory-wide refinement. In closed-loop control, the refined plan is repeatedly re-anchored by fresh observations rather than optimized once and executed open loop. By removing recursive decoded-state feedback within each span, PPWM targets both long-horizon predictive accuracy and planning efficiency: it removes one recursive error-propagation path and shortens the sequential dependency exposed to planning. We evaluate these effects through prediction accuracy, simulator planning, closed-loop refinement, architectural generality, and wall-clock scaling.

Our contributions are:

- **Parallel Predictive World Models.** We introduce PPWM, which replaces state-by-state decoded prediction with parallel causal trajectory prediction while preserving interaction among future representations.
- **Causal trajectory factorization for planning.** We characterize autoregressive rollout as a causal finite-horizon map and remove its decoded-state feedback pathway, reducing sequential model evaluation while exposing the full trajectory to planning.
- **Accurate and efficient long-horizon planning.** Across four visual-control tasks, PPWM achieves the lowest long-horizon prediction error and strongest CEM simulator performance, with over $3\times$ lower average planning latency than autoregressive LeWM.

2 Related Work

World models for planning. World models support control by coupling learned dynamics with test-time optimization. PlaNet plans in latent space from pixels (Hafner et al., 2019). TD-MPC and TD-MPC2 combine learned dynamics with local trajectory optimization (Hansen et al., 2022; 2024), while IRIS, DINO-WM, PLDM, and LeWorldModel extend planning to expressive sequence models,

offline visual data, and predictive representations (Micheli et al., 2022; Zhou et al., 2024; Sobal et al., 2026; Maes et al., 2026). However, most of these models still expose long-horizon futures through recursively composed local transitions, making both prediction and planning increasingly dependent on rollout depth.

Multi-step and parallel predictive dynamics. Long-horizon prediction can be improved either through multi-step objectives or by changing the predictive interface. Latent overshooting improves consistency while retaining recursive dynamics (Hafner et al., 2019), whereas Temporal Segment Models directly parameterize future trajectory segments (Mishra et al., 2017). Direct multi-step prediction can reduce recursive bias under partial observability or misspecification, while well-specified one-step models may remain statistically preferable (Somalwar et al., 2025; 2026). Most closely related, FAST-LEWM accumulates action effects with causal action prefixes and predicts multiple horizons in parallel (Gao and Xu, 2026). Existing approaches therefore either retain recursive dynamics or remove state recursion without explicitly modeling causal interaction among future representations.

Planning and trajectory optimization with learned world models. Sampling-based methods such as CEM, MPPI, and PETS optimize actions through repeated model evaluation (De Boer et al., 2005; Williams et al., 2017; Chua et al., 2018), while differentiable planners exploit learned-model geometry or trajectory-level objectives for local refinement (Georgiev et al., 2025; Pham and Bera, 2026; Huang et al., 2026). Lifted methods instead optimize intermediate states explicitly: LatCo applies latent-space collocation (Rybkin et al., 2021), GRASP exposes parallel optimization with virtual states (Psenka et al., 2026), and GVP-WM jointly optimizes latent states and actions (Ziakas et al., 2026). Related control work uses direct multi-step prediction in structured state–control optimization (Wu et al., 2026). These planners improve trajectory optimization, but their computation and optimization structure remain constrained by the temporal factorization exposed by the underlying world model.

3 PROBLEM FORMULATION

We consider goal-conditioned control from visual observations. Let $\boldsymbol{o}_t$ denote the observation, $\boldsymbol{a}_t \in \mathbb{A}$ the action, and $\boldsymbol{z}_t = E(\boldsymbol{o}_t) \in \mathbb{R}^D$ the learned representation used for prediction and planning. We denote by $\mathcal{C}_t$ the context available to the world model at time t, such as recent latent states and executed actions. Given a goal latent $\boldsymbol{z}_g = E(\boldsymbol{o}_g)$ and a planning horizon H, let

$$\boldsymbol{A} = (\boldsymbol{a}_t, \dots, \boldsymbol{a}_{t+H-1}), \qquad \boldsymbol{Z} = (\boldsymbol{z}_{t+1}, \dots, \boldsymbol{z}_{t+H}) \tag{1}$$

denote a candidate action sequence and its corresponding future trajectory. A world model predicts

$$\widehat{\boldsymbol{Z}} = \mathcal{P}_\theta(\mathcal{C}_t, \boldsymbol{A}), \tag{2}$$

and the planner seeks

$$\boldsymbol{A}^\star = \arg\min_{\boldsymbol{A} \in \mathbb{A}^H} J_{\text{task}}\left(\mathcal{P}_\theta(\mathcal{C}_t, \boldsymbol{A}), \boldsymbol{A}; \boldsymbol{z}_g\right). \tag{3}$$

A conventional autoregressive world model predicts future states recursively. Longer-horizon predictions are obtained by repeatedly feeding predicted states back into the model,

$$\widehat{\boldsymbol{z}}_{t+j+1} = f_\theta(\widehat{\boldsymbol{z}}_{t+j}, \boldsymbol{a}_{t+j}), \qquad \widehat{\boldsymbol{z}}_t = \boldsymbol{z}_t, \qquad j = 0, \dots, H-1. \tag{4}$$

This recursive rollout has two important consequences for long-horizon planning. First, predicted states are repeatedly reused as inputs, so local prediction errors can propagate over the rollout. Second, each future state depends on the preceding prediction, forming a chain of H sequentially dependent transition steps and limiting parallelism within each trajectory.

4 PARALLEL PREDICTIVE WORLD MODELS FOR LONG-HORIZON PLANNING

PPWM is designed for long-horizon prediction and planning by exposing a causally structured finite-horizon trajectory in a single forward pass, avoiding recursive decoded-state feedback and the horizon-length sequential dependency of autoregressive rollout. Figure 1 compares this design with

autoregressive rollout and Fast-LeWM-style prefix-conditioned direct prediction. Each prediction horizon is represented by a query conditioned on its causal action prefix, and the future queries interact through a shared causal trajectory predictor. The same finite-horizon interface exposes all future variables jointly, which also enables lightweight trajectory-wide refinement.

4.1 Parallel Future-State Prediction

Given the current context $\mathcal{C}_t$ and an action span of length K, PPWM predicts

$$\Phi_\theta(\mathcal{C}_t, \boldsymbol{A}_{t:t+K-1}) = (\widehat{\boldsymbol{z}}_{t+1}, \ldots, \widehat{\boldsymbol{z}}_{t+K}) \,. \tag{5}$$

All K future states are produced in a single forward pass. Unlike autoregressive rollout, $\widehat{\boldsymbol{z}}_{t+k}$ is not fed back as the input used to generate $\widehat{\boldsymbol{z}}_{t+k+1}$. Instead, each horizon is conditioned on the actions that causally precede it, while temporal dependence among future states is retained through hidden representations.

For each future horizon, a shared causal prefix encoder produces an action-prefix representation

$$(\boldsymbol{p}_{t+1}, \ldots, \boldsymbol{p}_{t+K}) = E_\psi(\mathcal{C}_t, \boldsymbol{a}_{t:t+K-1}; \boldsymbol{M}_{\text{causal}}) \,, \tag{6}$$

where $\boldsymbol{p}_{t+k}$ depends only on the current context and actions up to $\boldsymbol{a}_{t+k-1}$. We then construct

$$\boldsymbol{q}_1 = \boldsymbol{z}_t, \qquad \boldsymbol{q}_k = \boldsymbol{z}_t + \boldsymbol{e}_k + \boldsymbol{W}_p \boldsymbol{p}_{t+k}, \qquad k = 2, \ldots, K, \tag{7}$$

where $\boldsymbol{e}_k$ identifies the prediction horizon and $\boldsymbol{W}_p \boldsymbol{p}_{t+k}$ represents the accumulated effect of the causal action prefix.

4.2 Causal Trajectory Construction

The future queries are arranged in temporal order and jointly processed by a shared causal trajectory predictor. Let $\boldsymbol{u}_{t+k}^{(0)} = \boldsymbol{q}_k$ denote the input representation at horizon k. The predictor contains L causal Transformer layers, and each layer updates the representation at horizon k using only the current and preceding future positions.

$$\boldsymbol{u}_{t+k}^{(\ell)} = \mathcal{B}_\theta^{(\ell)} \left(\mathcal{C}_t, \boldsymbol{u}_{t+1:t+k}^{(\ell-1)}, \boldsymbol{A}_{t:t+k-1} \right), \qquad k = 1, \ldots, K, \tag{8}$$

where $\mathcal{B}_\theta^{(\ell)}$ denotes the ℓ-th causal Transformer block. The causal structure allows information from earlier future horizons to be progressively propagated and refined through the Transformer layers, while preventing access to later future positions or actions.

After the final layer, each future state is decoded from its corresponding representation.

$$\widehat{\boldsymbol{z}}_{t+k} = F_\theta \left(\boldsymbol{u}_{t+k}^{(L)} \right), \qquad k = 1, \ldots, K. \tag{9}$$

The decoded state $\widehat{\boldsymbol{z}}_{t+k}$ is never fed back into the prediction of later horizons, so temporal dependence is carried through the causal hidden representations rather than through recursive state prediction.

This factorization distinguishes PPWM from Fast-LeWM-style prefix-conditioned direct prediction. As illustrated in Figure 1, both approaches remove recursive feedback of predicted states, but PPWM additionally preserves explicit interaction among future-state representations before decoding. PPWM therefore replaces autoregressive state feedback with causal interaction in latent space, retaining temporal structure across the predicted trajectory without restoring serial state recursion.

4.3 Self-Conditioned Long-Horizon Composition

As our focus is the parallel predictive dynamics, we formulate the training objective directly in latent space and treat representation learning as orthogonal. All outputs within a span are directly supervised by their corresponding future latent targets,

$$\mathcal{L}_{\text{direct}} = \frac{1}{KD} \sum_{k=1}^{K} \left\| \widehat{\boldsymbol{z}}_{t+k} - \boldsymbol{z}_{t+k}^{\text{GT}} \right\|_2^2 . \tag{10}$$

For horizons longer than K, multiple PPWM spans are composed, so later spans receive model-generated rather than observed contexts. Training only from observed contexts would therefore create a mismatch between single-span training and long-horizon prediction. We address this with multi-anchor self-conditioned rollout (MAR) training. For a sampled restart anchor s, let $\widehat{\mathcal{C}}_{t+s}$ denote the detached predicted context and let R_s be the remaining supervised horizon. We define

$$\mathcal{L}_{\mathrm{MAR}} = \mathbb{E}_s \left[\frac{1}{R_s D} \sum_{j=1}^{R_s} \left\| \Phi_{\theta,j}\left(\mathrm{sg}[\widehat{\mathcal{C}}_{t+s}], \boldsymbol{A}_{t+s:t+s+R_s-1} \right) - \boldsymbol{z}^{\mathrm{GT}}_{t+s+j} \right\|_2^2 \right], \tag{11}$$

where $\Phi_{\theta,j}$ denotes the j-th prediction of the restarted span and $\mathrm{sg}[\cdot]$ denotes the stop-gradient operator. The final training objective is

$$\mathcal{L}_{\mathrm{PPWM}} = \mathcal{L}_{\mathrm{direct}} + \lambda_{\mathrm{MAR}} \mathcal{L}_{\mathrm{MAR}}. \tag{12}$$

The same finite-span predictor is therefore trained on the model-generated contexts encountered when spans are composed at test time. For a horizon $H > K$, prediction remains parallel within each span and becomes sequential only across span boundaries.

4.4 Parallel Trajectory Refinement

The same finite-horizon interface can also be differentiated at trajectory level. Because PPWM exposes the complete predicted trajectory $\widehat{\boldsymbol{Z}}(\boldsymbol{A}) = \Phi_\theta(\mathcal{C}_t, \boldsymbol{A})$ in one computational graph, the planner can optimize all future action and latent variables jointly instead of propagating updates through a decoded-state rollout. We use this property as a lightweight refinement of a plan produced by the Cross-Entropy Method (CEM).

Starting from the CEM plan $\boldsymbol{A}_0$, we introduce an auxiliary latent trajectory $\boldsymbol{Z}$ and initialize

$$\boldsymbol{A}^{(0)} = \boldsymbol{A}_0, \qquad \boldsymbol{Z}^{(0)} = \widehat{\boldsymbol{Z}}(\boldsymbol{A}_0). \tag{13}$$

The refinement couples the free latent trajectory to the trajectory induced by the current actions through

$$\mathcal{L}_{\mathrm{ref}}(\boldsymbol{A}, \boldsymbol{Z}) = J_{\mathrm{task}}(\boldsymbol{Z}) + \alpha J_{\mathrm{task}}(\widehat{\boldsymbol{Z}}(\boldsymbol{A})) + \lambda_{\mathrm{dyn}} \|\boldsymbol{Z} - \widehat{\boldsymbol{Z}}(\boldsymbol{A})\|_F^2 + \lambda_A \|\boldsymbol{A} - \boldsymbol{A}_0\|_F^2. \tag{14}$$

The first term is the task objective evaluated on the latent trajectory $\boldsymbol{Z}$, providing a direct optimization path toward the task goal in trajectory space. The executable-trajectory and consistency terms keep the update tied to model-predicted dynamics, while the final term regularizes actions around the CEM solution.

At refinement step m, gradients with respect to the complete action and latent trajectories are computed from the same objective and updated jointly,

$$\boldsymbol{A}^{(m+1)} = \Pi_{\mathcal{B}}\left(\boldsymbol{A}^{(m)} - \eta_A \nabla_{\boldsymbol{A}} \mathcal{L}_{\mathrm{ref}}\left(\boldsymbol{A}^{(m)}, \boldsymbol{Z}^{(m)} \right) \right), \boldsymbol{Z}^{(m+1)} = \boldsymbol{Z}^{(m)} - \eta_Z \nabla_{\boldsymbol{Z}} \mathcal{L}_{\mathrm{ref}}\left(\boldsymbol{A}^{(m)}, \boldsymbol{Z}^{(m)} \right). \tag{15}$$

where $\Pi_{\mathcal{B}}$ enforces a local feasible action region $\mathcal{B}$ around the CEM plan. Since PPWM produces all future positions within the span jointly, each refinement step updates the full action and latent trajectories without restoring a decoded-state rollout. After a small number of refinement steps, the action candidates are re-evaluated through PPWM and selected using the executable model trajectory,

$$\boldsymbol{A}^\star = \arg \min_{\boldsymbol{A} \in \{\boldsymbol{A}_0, \boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(R)}\}} J_{\mathrm{task}}\left(\widehat{\boldsymbol{Z}}(\boldsymbol{A}) \right). \tag{16}$$

Keeping $\boldsymbol{A}_0$ in the candidate set provides a fallback under the fixed model score.

5 Analysis of Temporal Factorization

PPWM preserves temporal dependence while changing its computational factorization. We first show that a finite autoregressive rollout is itself a causal trajectory map, so decoded-state recursion is a computational factorization rather than a requirement for temporal causality. We then characterize the decoded-state feedback pathway that is present in autoregressive rollout but absent from PPWM, while causal interaction among future hidden representations is retained.

5.1 AUTOREGRESSIVE ROLLOUT AS A CAUSAL TRAJECTORY MAP

Consider a deterministic autoregressive predictor over a finite horizon K,

$$\widehat{\boldsymbol{z}}_{t+1} = g_1(\mathcal{C}_t, \boldsymbol{a}_t), \qquad \widehat{\boldsymbol{z}}_{t+k} = g_k(\mathcal{C}_t, \widehat{\boldsymbol{z}}_{t+1:t+k-1}, \boldsymbol{a}_{t:t+k-1}), \quad k = 2, \dots, K. \tag{17}$$

Although this computation is expressed recursively, its finite-horizon input–output map need not be written in recursive form.

Theorem 1 (Equivalent causal trajectory factorization)**.** *For every finite deterministic autoregressive predictor of the form in Eq. (17), there exist causal functions $G_1, \dots, G_K$ such that*

$$\widehat{\boldsymbol{z}}_{t+k} = G_k(\mathcal{C}_t, \boldsymbol{a}_{t:t+k-1}), \qquad k = 1, \dots, K. \tag{18}$$

Hence the complete autoregressive rollout is exactly a causal finite-horizon map from the current context and action sequence to the predicted trajectory.

Theorem 1 separates temporal causality from a particular computational factorization. It shows that decoded-state recursion is not required by the causal input–output structure itself. PPWM realizes this structure while carrying cross-horizon dependence through causal hidden representations.

5.2 DECODED-STATE FEEDBACK PROPAGATION

The equivalent input–output form in Theorem 1 does not imply an equivalent computational graph. For the canonical autoregressive transition

$$\widehat{\boldsymbol{z}}_{t+m+1} = f_{\theta,m}(\widehat{\boldsymbol{z}}_{t+m}, \boldsymbol{a}_{t+m}), \tag{19}$$

every decoded prediction is reused as the input to the next transition. PPWM instead performs causal interaction among future hidden representations and decodes each future latent only after this interaction.

Theorem 2 (Elimination of recursive decoded-state feedback)**.** *Assume the autoregressive transitions in Eq. (19) are differentiable, and fix the action sequence. If the decoded state at horizon j is perturbed as $\widehat{\boldsymbol{z}}_{t+j} \leftarrow \widehat{\boldsymbol{z}}_{t+j} + \delta$, then for any $k > j$ the induced autoregressive perturbation satisfies*

$$\Delta\widehat{\boldsymbol{z}}_{t+k} = J_{k-1} \cdots J_j \, \delta + o(\|\delta\|), \qquad J_m = \frac{\partial f_{\theta,m}}{\partial \boldsymbol{z}}\Big|_{(\widehat{\boldsymbol{z}}_{t+m}, \boldsymbol{a}_{t+m})}. \tag{20}$$

Within a PPWM prediction span, if an additive perturbation is applied only to the decoded output node $\widehat{\boldsymbol{z}}_{t+j}$ after all hidden future representations have been computed, then every later decoded output $\widehat{\boldsymbol{z}}_{t+k}$, $k > j$, is unchanged.

Theorem 2 isolates one structural error-propagation path: autoregressive rollout contains a directed decoded-state feedback chain whose local sensitivities multiply with horizon, whereas PPWM has no such path within a span. Hidden-state perturbations may still propagate through Eq. (8). This retained causal interaction is precisely what distinguishes PPWM from independent horizon-wise prediction.

6 EXPERIMENTS

We evaluate PPWM from predictive modeling to downstream planning through five questions:

- **Q1: Predictive fidelity.** Does parallel causal prediction improve long-horizon accuracy?
- **Q2: Planning utility.** Does the PPWM interface improve simulator planning?
- **Q3: Planning efficiency.** How do prediction and planning costs scale with horizon?
- **Q4: Local refinement.** Can the parallel trajectory interface support effective refinement with feedback?
- **Q5: Architectural generality.** Does the same parallel prediction mechanism remain effective on different world-model architectures?

Experiments use Two-Room, Reacher, Push-T, and OGBench-Cube (Maes et al., 2026; Gao and Xu, 2026). LeWM-based models provide the main controlled comparison, while DINO-WM is used to evaluate architectural transfer. Planning success is measured from simulator execution. Detailed protocols, hyperparameter settings, and additional ablations are provided in Appendices A–D.

Table 1: Controlled prediction in the shared frozen LeWM latent space. Mean MSE over the first 5, 10, or 15 prediction steps (lower is better).

	OGBench-Cube			Push-T			Two-Room			Reacher		
Model	$\bar{E}_{1:5}$	$\bar{E}_{1:10}$	$\bar{E}_{1:15}$	$\bar{E}_{1:5}$	$\bar{E}_{1:10}$	$\bar{E}_{1:15}$	$\bar{E}_{1:5}$	$\bar{E}_{1:10}$	$\bar{E}_{1:15}$	$\bar{E}_{1:5}$	$\bar{E}_{1:10}$	$\bar{E}_{1:15}$
TD-MPC2	0.0310	0.0452	0.0525	0.0511	0.0888	0.1454	0.2126	0.2601	0.2951	0.0628	0.0670	0.0703
PLDM	0.0200	0.0325	0.0374	0.0371	0.0740	0.1319	0.0749	0.1153	0.1480	0.0557	0.0586	0.0605
LeWM	0.0173	0.0343	0.0456	0.0312	0.0967	0.1852	0.1139	0.1867	0.2707	0.0133	0.0199	0.0266
Fast-LeWM	0.0144	0.0268	0.0319	0.0189	0.0526	0.1047	0.0536	0.1106	0.1610	0.0520	0.0555	0.0728
PP-LeWM	**0.0130**	**0.0222**	**0.0270**	**0.0187**	**0.0485**	**0.0949**	**0.0395**	**0.0780**	**0.1159**	**0.0007**	**0.0009**	**0.0011**

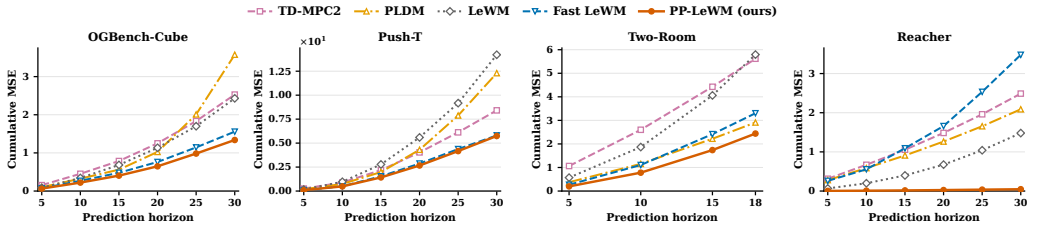

Figure 2: Cumulative latent prediction MSE as the open-loop prediction horizon increases. All methods use the same frozen LeWM representation (lower is better).

6.1 Long-Horizon Prediction

We first test whether PPWM improves long-horizon predictive accuracy. All methods use the same frozen LeWM encoder and projector, placing predictions and targets in a shared $D = 192$ latent space and making raw latent MSE directly comparable. We compare the native autoregressive LeWM predictor, recurrent and feed-forward one-step adaptations inspired by PLDM and TD-MPC2 (Sobal et al., 2026; Hansen et al., 2024), Fast-LeWM (Gao and Xu, 2026), and PP-LeWM. PP-LeWM instantiates PPWM on the LeWM architecture by retaining its representation and conditional Transformer backbone while replacing autoregressive future-state prediction with the proposed parallel causal trajectory interface.

Table 1 shows that PP-LeWM achieves the lowest MSE at all reported horizons on all four tasks. Fast-LeWM is the closest direct parallel baseline. It removes decoded-state recursion through action-prefix-conditioned prediction, whereas PP-LeWM additionally retains causal interaction among future representations before decoding. Because Fast-LeWM already removes decoded-state recursion, the consistent gap points to cross-horizon interaction as a useful design choice inside a parallel predictor. Reacher shows the largest separation. Its smooth, low-dimensional dynamics provide a regime in which cross-horizon structure can be exploited consistently across the prediction span. The cumulative-error curves in Figure 2 retain the same ordering beyond the tabulated horizons. PP-LeWM exhibits the slowest cumulative error growth across all four tasks, while the gap to autoregressive LeWM and Fast-LeWM generally widens at longer horizons. The separation grows most clearly on Reacher and Two-Room, where PP-LeWM accumulates substantially less error as prediction moves farther from the observed context.

6.2 Planning Across Predictive Interfaces

We evaluate whether the predictive differences persist after coupling each model to downstream planners. We compare LeWM, Fast-LeWM, and PP-LeWM under trajectory-wide LatCo and GRASP optimization as well as population-based CEM planning. Each completed trial plans over a horizon of 50 primitive simulator actions, and executes the full plan open loop. Anytime and Terminal success are computed only from the resulting simulator trajectory. Table 2 summarizes the resulting open-loop success rates across planners and predictive interfaces.

Under CEM, PP-LeWM achieves the highest anytime and terminal success on all four tasks. The lifted planners expose a different model–planner interaction: Fast-LeWM remains competitive on Cube and Two-Room but degrades sharply with LatCo on Push-T and Reacher, while GRASP exhibits

Table 2: Open-loop simulator success across predictive interfaces (100 episodes × 3 planner seeds; higher is better). CEM uses each model's frozen scoring rule; see Appendix C.3.

		OGBench-Cube		Push-T		Two-Room		Reacher	
Planner	Model	Anytime↑	Terminal↑	Anytime↑	Terminal↑	Anytime↑	Terminal↑	Anytime↑	Terminal↑
LatCo	LeWM	28.67	20.67	0.67	0.33	3.33	1.67	35.67	18.67
	Fast-LeWM	31.00	22.33	1.67	0.67	7.67	5.67	15.67	2.33
	PP-LeWM	31.00	26.00	16.67	12.67	8.67	5.33	79.33	78.67
GRASP	LeWM	31.33	21.67	9.00	0.33	9.33	5.00	81.00	11.33
	Fast-LeWM	29.00	18.67	15.00	2.00	24.00	20.00	62.67	5.67
	PP-LeWM	34.00	26.00	17.00	3.67	10.33	4.67	38.67	0.67
CEM	LeWM	35.00	28.67	35.33	27.67	49.67	49.67	76.67	64.33
	Fast-LeWM	38.00	35.33	35.33	28.67	44.67	44.33	51.33	35.00
	PP-LeWM	**45.67**	**39.33**	**37.67**	**29.33**	**55.33**	**55.00**	**97.00**	**96.67**

Table 3: CEM planning latency at $H=10$ (ms); speedup relative to LeWM is shown below.

Dataset	LeWM	Fast-LeWM	PP-LeWM
OGBench-Cube	1245.4	474.3	**386.7**
	1.00×	2.63×	3.22×
Push-T	1252.9	457.7	**357.0**
	1.00×	2.74×	3.51×
Two-Room	1243.8	505.9	**385.6**
	1.00×	2.46×	3.23×
Reacher	1198.6	465.3	**368.9**
	1.00×	2.58×	3.25×
Average	1235.2	475.8	**374.5**
	1.00×	2.60×	3.30×

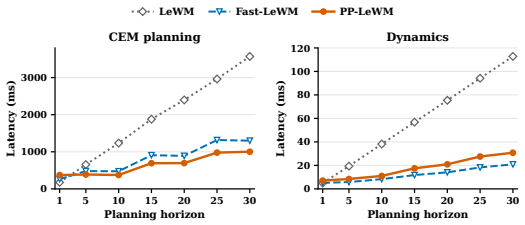

Figure 3: Average CEM planning and dynamics latency across four tasks.

high anytime but low terminal success on Reacher. CEM evaluates Fast-LeWM through its native direct finite-horizon interface, whereas the LatCo and GRASP adaptations require a chained one-step dynamics factor over optimized latent states. Appendix D.1 shows large virtual-to-rollout gaps when Fast-LeWM is forced into this chained local factor. The mismatch is therefore an interface issue: lifted optimization can improve the local surrogate without improving the native finite-horizon prediction used by the direct model.

6.3 Planning Efficiency and Horizon Scaling

Planning efficiency is evaluated by measuring how dynamics and end-to-end CEM latency scale. Table 3 summarizes CEM planning latency at a ten-step model horizon, while Figure 3 reports average CEM planning and dynamics latency for planning horizons of 1, 5, 10, 15, 20, 25, and 30 steps. PP-LeWM achieves the lowest CEM planning latency on all four tasks, averaging 374.5 ms compared with 1235.2 ms for LeWM and 475.8 ms for Fast-LeWM. This corresponds to a 3.30× average speedup over LeWM. Figure 3 further shows that LeWM latency grows rapidly with rollout length, whereas PP-LeWM exhibits span-wise growth and remains substantially more efficient at medium and long horizons. Fast-LeWM has the lowest raw dynamics latency due to its lighter direct predictor, but its re-anchored self-consistency score adds planner-side model evaluations. Score-level details are provided in Appendix D.3. PP-LeWM therefore delivers the lowest end-to-end CEM latency while retaining causal interaction among future representations.

6.4 Closed-Loop Trajectory Refinement

To evaluate trajectory-level refinement under feedback, we run PP-LeWM in closed loop on Push-T and Two-Room. The planner replans every five primitive simulator steps. At each decision, the agent receives a fresh observation, runs CEM over the remaining episode horizon, applies the local refinement, executes the first five primitive actions, and replans. We compare the unrefined CEM plan with action-only Adam and trajectory-wide refinement, which jointly updates the action sequence and an auxiliary latent trajectory. All Anytime and Terminal success rates are computed exclusively from the resulting closed-loop simulator trajectories, as shown in Table 4.

Table 4: Closed-loop refinement with PP-LeWM under a shared CEM planner (higher is better).

	Push-T		Two-Room	
Method	Anytime↑	Terminal↑	Anytime↑	Terminal↑
CEM	56.33	49.00	55.33	55.00
+ Adam	57.00	**50.33**	56.00	56.00
+ Traj. Refine	**58.33**	50.00	**58.00**	**57.67**

Table 5: PP-DINO-WM transfer on Push-T. DINOv2 visual-patch MSE (lower is better).

Prediction Steps	DINO-WM	PP-DINO-WM	Reduction
5	0.098855	**0.080229**	18.84%
10	0.154497	**0.108464**	29.80%
15	0.219716	**0.147109**	33.05%
20	0.284458	**0.180760**	36.45%

Table 6: Push-T component analysis using mean latent MSE; lower is better.

Model	$\bar{E}_{1:10} \downarrow$	$\bar{E}_{1:15} \downarrow$	$\bar{E}_{1:20} \downarrow$	$\bar{E}_{1:30} \downarrow$
Fast-LeWM	0.052634	0.104680	0.146183	0.208614
PP-LeWM w/o MAR	0.051014	0.102552	0.150012	0.239441
PP-LeWM	**0.048525**	**0.094867**	**0.133276**	**0.191456**

Trajectory refinement provides the clearest closed-loop gain on Two-Room, improving both Anytime and Terminal success by 2.67 percentage points over CEM and consistently outperforming action-only Adam. The additional gain over action-only Adam on Two-Room comes with the ability to move the latent trajectory and actions together across the full horizon. Moreover, the parallel trajectory interface reduces the sequential prediction depth of each refinement iteration from $\mathcal{O}(H)$ autoregressive transitions to $\mathcal{O}(\lceil H/K \rceil)$ PPWM span evaluations, and to $\mathcal{O}(1)$ when $H \leq K$. Closed-loop replanning therefore combines trajectory-wide updates with a short sequential optimization path while repeatedly re-anchoring the model to observed states.

6.5 Architectural Generality on DINO-WM

To test architectural transfer, we instantiate PPWM on DINO-WM (Zhou et al., 2024). PP-DINO-WM retains the frozen DINOv2 representation (Oquab et al., 2023) while replacing autoregressive future prediction with the proposed parallel causal trajectory interface. The autoregressive DINO-WM and PP-DINO-WM use the same test windows, histories, and future actions and are rolled out open loop. Table 5 reports prediction error across the evaluated horizons. PP-DINO-WM reduces visual latent error at every reported horizon, with the relative reduction increasing from 18.84% over 5 steps to 36.45% over 20 steps. The same temporal factorization therefore remains effective after replacing the LeWM backbone with DINO-WM.

6.6 PPWM Component Analysis

Table 6 isolates the two main components of PP-LeWM on Push-T. Fast-LeWM provides a parallel reference without causal interaction among future representations, while PP-LeWM w/o MAR removes multi-anchor rollout training. Full PP-LeWM achieves the lowest error at every horizon. These results show that causal trajectory construction and MAR are both important for maintaining accurate long-horizon prediction.

7 Conclusion

We introduced Parallel Predictive World Models (PPWM) for accurate and efficient long-horizon planning. PPWM replaces recursive decoded-state prediction with parallel causal trajectory prediction, preserving temporal interaction among future representations while exposing the finite-horizon trajectory jointly to planning. Across four visual-control tasks, PPWM consistently improves long-horizon prediction and CEM planning performance while achieving over a $3\times$ average planning speedup relative to autoregressive LeWM. The same factorization transfers to DINO-WM and naturally supports trajectory-wide refinement, demonstrating its compatibility with different world-model backbones and planning regimes. More broadly, PPWM shows that long-horizon world models can preserve causal temporal structure without relying on state-by-state autoregressive rollout, providing a direct and efficient alternative for trajectory prediction and planning.

AI USE STATEMENT

Generative AI tools were used for language editing, manuscript organization, literature search, reference verification, and notation and consistency checking. All AI-assisted suggestions and bibliographic information were manually verified by the authors. The authors take full responsibility for the final manuscript.

REPRODUCIBILITY STATEMENT

Sections 3–6 define PPWM, its predictive and planning formulations, and the evaluation questions. Appendix A provides implementation details, Appendix B states assumptions and derivations, Appendix C specifies the experimental protocols and metric definitions, and Appendix D collects additional ablations and diagnostics. The code will be made publicly available upon acceptance.

REFERENCES

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

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

Ignat Georgiev, Varun Giridhar, Nick Hansen, and Animesh Garg. Pwm: Policy learning with multi-task world models. In *International Conference on Learning Representations*, volume 2025, pages 13737–13757, 2025.

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*, pages 2555–2565. PMLR, 2019.

Nick Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations*, volume 2024, pages 47376–47405, 2024.

Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. *arXiv preprint arXiv:2203.04955*, 2022.

Xiaodi Huang, Ziyi Ding, Jingtian Wan, Yuchen Liu, Yuan Zhang, Xiao-Ping Zhang, Jiayu Chen, Zhang Zhang, and Tao Huang. Traj-lewm: Path-aware world-model planning via latent trajectory cost. *arXiv preprint arXiv:2608.14125*, 2026.

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

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.

Vincent Micheli, Eloi Alonso, and François Fleuret. Transformers are sample-efficient world models. *arXiv preprint arXiv:2209.00588*, 2022.

Nikhil Mishra, Pieter Abbeel, and Igor Mordatch. Prediction and control with temporal segment models. In *International conference on machine learning*, pages 2459–2468. PMLR, 2017.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

Phu Pham and Aniket Bera. Latent energy action planning with world models. *arXiv preprint arXiv:2609.03294*, 2026.

Michael Psenka, Michael Rabbat, Aditi Krishnapriyan, Yann LeCun, and Amir Bar. Parallel stochastic gradient-based planning for world models. *arXiv preprint arXiv:2602.00475*, 2026.

Oleh Rybkin, Chuning Zhu, Anusha Nagabandi, Kostas Daniilidis, Igor Mordatch, and Sergey Levine. Model-based reinforcement learning via latent-space collocation. In *International Conference on Machine Learning*, pages 9190–9201. PMLR, 2021.

Uladzislau Sobal, Wancong Zhang, Kyunghyun Cho, Randall Balestriero, Tim GJ Rudner, and Yann LeCun. Learning from reward-free offline data: A case for planning with latent dynamics models. *Advances in Neural Information Processing Systems*, 38:43905–43941, 2026.

Anne Somalwar, Bruce D Lee, George J Pappas, and Nikolai Matni. Learning with imperfect models: When multi-step prediction mitigates compounding error. In *2025 IEEE 64th Conference on Decision and Control (CDC)*, pages 82–89. IEEE, 2025.

Anne Somalwar, Bruce D Lee, George J Pappas, and Nikolai Matni. Statistical efficiency of single-and multi-step models for forecasting and control. *arXiv preprint arXiv:2603.23465*, 2026.

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)*, pages 1714–1721. IEEE, 2017.

Liang Wu, Wallace Gian Yion Tan, Richard D Braatz, and Ján Drgoňa. Koopman-boxqp: Solving large-scale nmpc at khz rates. *arXiv preprint arXiv:2602.18331*, 2026.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. Dino-wm: World models on pre-trained visual features enable zero-shot planning. *arXiv preprint arXiv:2411.04983*, 2024.

Christos Ziakas, Amir Bar, and Alessandra Russo. Grounding generated videos in feasible plans via world models. *arXiv preprint arXiv:2602.01960*, 2026.

A Implementation Details

A.1 PP-LeWM instantiation

Representation, temporal scale, and context. The visual encoder and projector are frozen throughout PP-LeWM training. The latent dimension is $D = 192$. For Push-T and Reacher, one model-step action is the same five-primitive-action block used by LeWM. With two-dimensional primitive controls, the raw block dimension is 10. The main configuration uses $K = 10$ future model steps.

History context encoder. The action-prefix branch is conditioned on a compact motion context rather than only the latest latent. We compute

$$u_t = [z_t,\ z_t - z_{t-1},\ z_{t-1} - z_{t-2}] \in \mathbb{R}^{576} \tag{21}$$

and map it with a lightweight MLP

$$576 \to 384 \to 192 \tag{22}$$

using normalization and GELU between linear layers. Its output $h_t \in \mathbb{R}^{192}$ is used only by the action-prefix encoder. The original three-state LeWM trajectory input is retained separately.

History-conditioned action-prefix encoder. Each future action block is first projected to 32 dimensions and then mapped through a $32 \to 768 \to 192$ MLP. The history context h_t is separately mapped through a $192 \to 768 \to 192$ state-token MLP. We prepend this state token to the sequence of future action tokens, add fixed sinusoidal positional encodings, and process the sequence with a three-layer causal Transformer with model dimension 192, six heads, and FFN dimension 768. Dropping the context-token output yields $p_{t+1:t+K} \in \mathbb{R}^{K\times 192}$. A LayerNorm followed by a linear $192 \to 192$ prefix projection W_p maps these features into the future-query space. The projection is initialized near zero so that the new prefix branch initially perturbs the pretrained trajectory dynamics only weakly.

Horizon-aware future queries. Fixed horizon codes $e_k \in \mathbb{R}^{192}$ are added to the future-query tokens. We preserve the one-step LeWM prior by setting

$$q_1 = z_t, \qquad q_k = z_t + e_k + W_p p_{t+k}, \quad k = 2, \ldots, K. \tag{23}$$

The full trajectory sequence is therefore

$$[z_{t-2}, z_{t-1}, q_1, \ldots, q_K]. \tag{24}$$

The causal mask permits hidden trajectory interaction from earlier to later query tokens, but no predicted latent $\hat{z}_{t+k}$ is fed back as the input used to generate $\hat{z}_{t+k+1}$ within the span.

Conditional dynamics backbone and prediction head. The base LeWM predictor contains six causal conditional Transformer blocks with hidden dimension 192, AdaLN-style local action conditioning, and a 2048-dimensional feed-forward sublayer. The main model fine-tunes all six predictor blocks, the final normalization, and the pretrained prediction projection. The encoder, projector, and LeWM action encoder remain frozen. Future hidden states are decoded directly by the fine-tuned pretrained prediction projection.

Training curriculum. New history/prefix modules are first warmed up with the dynamics backbone frozen and $\mathcal{L}_{\text{direct}}$ only. We then jointly fine-tune the full dynamics backbone using a smaller learning rate for pretrained dynamics parameters than for the new history/prefix modules. After direct prediction stabilizes, MAR is enabled. During development, grouped learning rates are selected from 10^{-4}–3×10^{-4} for new modules and 10^{-5}–3×10^{-5} for pretrained dynamics parameters.

Direct and multi-anchor rollout losses. All K direct outputs receive equal-weight latent MSE supervision. For MAR, after the first span we choose two random anchors $s_1, s_2 \sim U(3, K-1)$ and the boundary anchor K. At every anchor, the three predicted states are detached before the next model call. The history encoder and prefix encoder are then recomputed from this detached predicted history, so the rollout objective trains the new action representation under self-generated dynamics context without backpropagating through the first-span predicted history. The final objective is

$$\mathcal{L}_{\text{PPWM}} = \mathcal{L}_{\text{direct}} + 0.25\, \mathcal{L}_{\text{MAR}}. \tag{25}$$

The base configuration uses no auxiliary task-specific losses. Nominal and action-perturbed trajectories share the same objective.

A.2 Controlled dynamics baselines

The controlled PLDM and TD-MPC2 baselines use the same cached LeWM latents and frozen action embedding as the other dynamics models, so the comparison isolates the temporal predictor. We retain their characteristic recurrent or feed-forward transition structures while omitting agent-specific components outside the controlled dynamics comparison.

PLDM. PLDM uses a recurrent predictor in its Two-Room instantiation. The original paper reports a two-layer GRU with 512 hidden dimensions (Sobal et al., 2026). Our controlled PLDM-GRU baseline projects the current 192-dimensional LeWM latent into the GRU hidden state, consumes the frozen LeWM action embeddings recurrently, and maps each recurrent output back to the 192-dimensional LeWM latent space. It is rolled out autoregressively at evaluation time. This baseline tests whether recurrent memory alters the long-horizon behavior of one-step autoregressive prediction while keeping the visual representation fixed.

TD-MPC2. TD-MPC2 uses a feed-forward one-step latent transition of the form $z_{t+1} = d_\theta(z_t, a_t)$ implemented as an MLP (Hansen et al., 2024). Our controlled adaptation concatenates the frozen LeWM latent and action embedding, predicts the next LeWM latent with an MLP transition, and recursively feeds predictions back at evaluation time.

Training control. The two adapted baselines are trained only on the cached LeWM latent/action trajectories used for the PPWM comparison. Their objectives supervise predicted LeWM latents and do not include the representation-learning losses of their source methods. Parameter counts, optimizer schedules, and exact hidden widths follow the controlled experiment configuration.

A.3 PP-DINO-WM INSTANTIATION

Representation and temporal scale. We instantiate PPWM on the Push-T DINO-WM backbone while keeping the DINOv2 ViT-S/14 visual encoder frozen. RGB observations are normalized to $[-1, 1]$ and resized to 196×196 before DINOv2 encoding. Each observation is represented by 196 spatial patch tokens of dimension 384. The four-dimensional proprioceptive state is linearly encoded to 10 dimensions and concatenated to every patch, giving a 394-dimensional state token. Histories contain three frames separated by five primitive environment steps. Five primitive two-dimensional actions are grouped into one 10-dimensional model-step action, and the native parallel span is $K = 10$, corresponding to 50 primitive environment steps.

Parallel dynamics. PP-DINO-WM uses the same PPWM factorization as Section 4: a causal action-prefix branch constructs horizon-specific future queries, which are jointly processed by the DINO-WM causal dynamics backbone. The prefix encoder has three Transformer layers with width 192, six attention heads, and FFN dimension 768. The DINO-WM dynamics backbone has six layers, 16 attention heads, and FFN dimension 2048. Attention is causal at the frame level, so each future frame can interact with preceding future-frame representations while remaining independent of later frames. All K future states are produced in one backbone call without feeding decoded states back within the span.

Initialization and training. The dynamics predictor, action encoder, and proprioceptive encoder are initialized from the official Push-T DINO-WM checkpoint, while the history/prefix branch is newly initialized. The episode-separated split contains 301,279 training windows, 512 development windows, and 1,008 test windows. The first five epochs train only the new branch. From epoch 6 onward, the dynamics predictor and action encoder are jointly fine-tuned. The results in Section 6.5 use the fixed epoch-10 checkpoint and therefore reflect direct 10-step supervision without MAR. Training uses four-GPU data parallelism and AdamW with learning rates of 10^{-4} for newly initialized parameters and 10^{-5} for pretrained dynamics parameters, weight decay 10^{-4}, gradient-norm clipping at 1, FP32 arithmetic, and an effective batch size of 128.

A.4 PARALLEL TRAJECTORY REFINEMENT DETAILS

CEM initialization and trajectory variables. At each closed-loop decision, refinement starts from the CEM action sequence $\boldsymbol{A}_0$ and its PPWM rollout $\widehat{\boldsymbol{Z}}(\boldsymbol{A}_0)$. The variable $\boldsymbol{Z}$ is auxiliary, while executable trajectories are always generated by the frozen PPWM.

Tail task cost. CEM and the trajectory refinement use the same normalized tail objective. For the remaining horizon H, let $q = \min(4, H)$ and define

$$J_{\text{task}} = J_\rho(\mathbf{Z}, \boldsymbol{z}_g) = \frac{\sum_{j=0}^{q-1} \rho^j \,\text{MSE}(\mathbf{z}_{H-j}, \boldsymbol{z}_g)}{\sum_{j=0}^{q-1} \rho^j}. \tag{26}$$

The terminal state receives the largest weight, and at most the final four model states contribute to the score. The refinement objective in Eq. (14) uses this score for both the free trajectory and the executable PPWM rollout, together with a soft dynamics-consistency penalty and a quadratic action penalty around the CEM reference plan.

B PROOFS AND ADDITIONAL ANALYSIS

B.1 PROOF OF THEOREM 1

We prove the claim by recursive substitution. For the first horizon, define

$$G_1(\mathcal{C}_t, \boldsymbol{a}_t) = g_1(\mathcal{C}_t, \boldsymbol{a}_t). \tag{27}$$

Then $\widehat{\boldsymbol{z}}_{t+1} = G_1(\mathcal{C}_t, \boldsymbol{a}_t)$. Assume that for every $j < k$ there exists a function G_j satisfying

$$\widehat{\boldsymbol{z}}_{t+j} = G_j(\mathcal{C}_t, \boldsymbol{a}_{t:t+j-1}). \tag{28}$$

Substituting these expressions into the k-th autoregressive prediction gives

$$G_k(\mathcal{C}_t, \boldsymbol{a}_{t:t+k-1}) = g_k(\mathcal{C}_t, G_1(\mathcal{C}_t, \boldsymbol{a}_t), \ldots, G_{k-1}(\mathcal{C}_t, \boldsymbol{a}_{t:t+k-2}), \boldsymbol{a}_{t:t+k-1})\,. \tag{29}$$

Therefore $\widehat{\boldsymbol{z}}_{t+k} = G_k(\mathcal{C}_t, \boldsymbol{a}_{t:t+k-1})$. By induction, this construction holds for every $k = 1, \ldots, K$. Each G_k depends only on the current context and the action prefix ending at $t+k-1$, so the resulting finite-horizon map is causal.

B.2 Proof of Theorem 2

Perturb $\widehat{\boldsymbol{z}}_{t+j}$ to $\widehat{\boldsymbol{z}}_{t+j} + \delta$ while keeping the action sequence fixed, and let Δ_m denote the induced perturbation of $\widehat{\boldsymbol{z}}_{t+m}$. Differentiability of Eq. (19) gives

$$\Delta_{m+1} = J_m \Delta_m + o(\|\Delta_m\|), \tag{30}$$

where J_m is defined in Eq. (20). Starting from $\Delta_j = \delta$ and recursively applying Eq. (30) yields

$$\Delta_k = J_{k-1} \cdots J_j \, \delta + o(\|\delta\|), \tag{31}$$

which proves the autoregressive statement. Hence the autoregressive computational graph contains a directed path from each decoded future state to every later decoded state.

For PPWM, the future hidden representations are computed by the causal trajectory blocks before decoding, as in Eq. (8), and each output is obtained by

$$\widehat{\boldsymbol{z}}_{t+k} = F_\theta(\boldsymbol{u}^{(L)}_{t+k}). \tag{32}$$

No decoded output $\widehat{\boldsymbol{z}}_{t+j}$ is an input to any later hidden representation $\boldsymbol{u}^{(\ell)}_{t+k}$. Therefore, after the hidden representations have been fixed, intervening on $\widehat{\boldsymbol{z}}_{t+j}$ after F_θ creates no directed computational path to any later output and induces exactly zero change in $\widehat{\boldsymbol{z}}_{t+k}$ for $k > j$. At the same time, causal hidden interaction remains because $\boldsymbol{u}^{(\ell)}_{t+k}$ can depend on earlier future representations $\boldsymbol{u}^{(\ell-1)}_{t+1:t+k}$. The result therefore removes recursive decoded-state feedback without asserting independence among PPWM predictions.

C Experimental Protocols

This appendix specifies the common evaluation protocol and the experiment-specific deviations used in the main text.

C.1 Evaluation Setup

One model step corresponds to five primitive simulator actions. Unless stated otherwise, open-loop planning uses a one-shot horizon of 10 model steps (50 primitive actions), and each configuration is evaluated on 100 episodes with three planner seeds. CEM uses 300 candidates, 30 elites, and 30 iterations in standardized action coordinates. Anytime success records whether the native task success condition is reached at any point during simulator execution, while Terminal success evaluates the final primitive step. Planner hyperparameters are selected on development data and frozen before test evaluation.

C.2 Controlled Prediction Protocol

The shared representation, baseline constructions, and training controls are described in Appendix A.2. All methods receive identical initial histories, future actions, and latent targets. For horizon j, we measure

$$E_j = \frac{1}{D} \left\| \hat{z}_{t+j} - z^{\mathrm{GT}}_{t+j} \right\|_2^2 . \tag{33}$$

The mean error over the first h steps and the cumulative error are

$$\bar{E}_{1:h} = \frac{1}{h} \sum_{j=1}^{h} E_j, \qquad C_h = \sum_{j=1}^{h} E_j = h \bar{E}_{1:h}. \tag{34}$$

Table 1 reports $\bar{E}_{1:h}$ and Figure 2 reports C_h.

C.3 Open-Loop CEM Protocol

The CEM rows in Table 2 follow the common setup above: each trial plans once from the observed three-frame history and goal, then executes the complete 50-step action sequence without replanning or early stopping. Push-T success requires the joint agent/block positional criterion and block-angle error below $\pi/9$. Reacher requires both absolute joint-position errors below 0.05 radians. Two-Room uses final agent-to-goal distance below 16 pixels, and OGBench-Cube uses final cube-to-goal distance at most 0.04 meters.

C.4 Open-Loop Trajectory Optimization Protocol

All planners use the same standardized action coordinates and environment bounds. LatCo uses its stochastic action initialization with start–goal state interpolation, while GRASP uses its lifted-state initialization with latent perturbations and periodic rollout synchronization. These initializations are kept separate from the CEM-initialized local refinement in Section 6.4.

C.5 Closed-Loop Refinement Protocol

The closed-loop study uses the common CEM search budget, but executes only one five-action block before receiving a new simulator observation and replanning. The remaining model horizon decreases from 10 to one over the 50-step episode, and both tasks use $\rho = 0.25$. All methods start each decision from the same CEM solution and preserve the unrefined plan as a fallback candidate. Adam refines only the standardized action sequence, whereas trajectory refinement jointly updates the action sequence and a free virtual latent trajectory. The frozen trajectory-refinement settings are

Parameter	Push-T	Two-Room
Rounds R	2	3
η_A	1	2
η_Z	24	48
α	1	1
λ_{dyn}	1	1
λ_A	0.01	0.01
Trust-region radius δ	1	0.25
Tail coefficient ρ	0.25	0.25

C.6 Single-GPU Timing Protocol

Timing uses horizons $H \in \{1, 5, 10, 15, 20, 25, 30\}$ on a single NVIDIA A100-SXM4-80GB GPU in FP32 evaluation mode. All methods receive the same three-frame latent history, past actions, and future action sequence. Data loading, image encoding, checkpoint loading, and simulator execution are excluded. Dynamics latency measures one evaluation of 300 candidate trajectories, and end-to-end CEM timing uses the common search budget from Appendix C.1. LeWM is rolled out autoregressively, while Fast-LeWM and PP-LeWM are composed in spans of at most 10 model steps. A complete CEM solve includes candidate sampling, action processing, dynamics prediction, objective evaluation, sorting, elite updates, and final plan scoring. CUDA synchronization is applied around timed regions, and the horizon-scaling curves report median wall-clock time.

C.7 DINO-WM Architectural Generality Protocol

The DINO-WM experiment evaluates both models on the same 1,008 held-out Push-T windows and known future action sequences. DINO-WM rolls out autoregressively, whereas PP-DINO-WM predicts 10 steps per span and composes two spans for the 20-step horizon. Prediction error is computed on the 196-by-384 DINOv2 visual patch features, averaging the stepwise MSE over all valid windows at each reported horizon.

D Additional Ablations and Diagnostics

This appendix contains three focused analyses that directly support the main results: a planner-interface diagnostic for the lifted Fast-LeWM comparisons, ablations of the PPWM parallel prediction span and self-conditioned long-horizon training, and a controlled analysis of the CEM scoring function.

D.1 Planner-Interface Diagnostic

Table 7: Planner-interface diagnostic for the Fast-LeWM lifted-planner adaptations. "Virtual" is the optimized terminal latent state, "chained rollout" re-evaluates the one-step planning factor from the true initial context, and "simulator" is the realized terminal latent MSE after executing the optimized actions. All entries use the completed 300-trial evaluations. Lower is better.

Task	Planner	Virtual MSE	Chained MSE	Simulator MSE
Push-T	LatCo	0.0036	1.5570	1.5449
Push-T	GRASP	1.0202	1.0155	1.2472
Reacher	LatCo	0.0002	0.9086	1.0736
Reacher	GRASP	0.1736	0.1752	0.6278

The large virtual-to-rollout gaps show that the chained one-step factor used by lifted Fast-LeWM can become misaligned with its native direct predictor, especially for LatCo. Fast-LeWM consequently performs well when CEM queries its native finite-horizon predictor, but degrades when lifted optimization replaces that interface with a chained local factor.

D.2 PPWM Component Ablations

We examine three design choices tied to the proposed parallel predictor: the native parallel prediction span K, causal trajectory construction, and multi-anchor self-conditioned rollout training (MAR).

Parallel prediction span. For the span study, $K \in \{5, 10, 15\}$ while all other predictor and rollout settings are held fixed.

Table 8: Effect of the native parallel prediction span K. Entries are mean latent MSE over the first h prediction steps, $\bar{E}_{1:h}$ (lower is better).

Dataset	K	$\bar{E}_{1:5} \downarrow$	$\bar{E}_{1:10} \downarrow$	$\bar{E}_{1:15} \downarrow$	$\bar{E}_{1:20} \downarrow$
Push-T	5	**0.017544**	0.050309	0.094625	0.138354
Push-T	10	0.018656	**0.048525**	0.094867	**0.133276**
Push-T	15	0.021414	0.053420	**0.093473**	0.143585
Reacher	5	0.000739	0.000926	0.001113	0.001296
Reacher	10	**0.000735**	**0.000900**	**0.001075**	**0.001261**
Reacher	15	0.000818	0.000959	0.001144	0.001345

$K = 10$ is best on all reported Reacher horizons and gives the lowest Push-T error at $h = 10$ and $h = 20$, so we use it as the default span.

D.3 CEM Score Controls and Tail Ablation

Table 2 reports end-to-end CEM performance under each model's frozen scoring rule. The controls below complement it with goal-only scoring and a PP-LeWM tail ablation.

CEM candidate scores. For candidate m, the ordinary terminal-goal score and Fast-LeWM self-consistency (SC) score (Gao and Xu, 2026) are

$$J_{\text{goal}}^{(m)} = \left\| \widehat{\boldsymbol{z}}_{t+H}^{(m)} - \boldsymbol{z}_g \right\|_2^2, \qquad J_{\text{Fast-SC}}^{(m)} = J_{\text{goal}}^{(m)} + \beta \left\| \widehat{\boldsymbol{z}}_{t+H}^{(m)} - \widetilde{\boldsymbol{z}}_{t+H}^{(m)} \right\|_2^2. \tag{35}$$

Here $\widetilde{\boldsymbol{z}}_{t+H}^{(m)}$ is Fast-LeWM's re-anchored terminal estimate, while PP-LeWM uses the tail objective J_ρ defined once in Eq. (26). Setting $\beta = 0$ recovers goal-only CEM, and $\rho = 0$ gives the same terminal-goal ranking for PP-LeWM.

Controlled open-loop comparison. Table 9 combines the CEM score control with the PP-LeWM tail sweep and reports simulator Anytime and Terminal success.

Table 9: Open-loop CEM score control and PP-LeWM tail ablation (H=10). Entries report Anytime and Terminal simulator success (%). Fast-LeWM Goal removes SC, while PP-LeWM $\rho = 0$ is ordinary terminal-goal CEM. Each configuration uses 100 episodes and three planner seeds.

		Push-T		Reacher		Two-Room		OGBench-Cube	
Model	CEM score	Anytime↑	Terminal↑	Anytime↑	Terminal↑	Anytime↑	Terminal↑	Anytime↑	Terminal↑
LeWM	Goal	35.33	27.67	76.67	64.33	49.67	49.67	35.00	28.67
Fast-LeWM	SC	35.33	28.67	51.33	35.00	44.67	44.33	38.00	35.33
Fast-LeWM	Goal	30.00	22.33	51.67	34.33	50.67	50.67	36.33	30.00
PP-LeWM	J_0 (Goal)	37.67	**29.33**	**97.00**	**96.67**	54.67	54.33	38.67	32.33
PP-LeWM	$J_{0.1}$	35.67	25.00	**97.00**	**96.67**	**55.33**	**55.00**	**45.67**	**39.33**
PP-LeWM	$J_{0.25}$	38.67	27.33	96.67	95.00	55.00	**55.00**	41.67	37.00
PP-LeWM	$J_{0.5}$	**41.33**	26.67	96.33	93.67	52.33	52.33	43.67	38.33

Under ordinary goal-only CEM (J_0), PP-LeWM already outperforms the goal-only LeWM and Fast-LeWM baselines on all four tasks, so the main planning gain does not depend on tail weighting. Tail weighting shifts the operating point by task: it raises success on OGBench-Cube and Two-Room and increases Push-T Anytime success, whereas Reacher already saturates under J_0. Fast-LeWM's SC term is similarly task dependent.