FAST PLANNING WITH CROSS-CANDIDATE INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Sampling-based planning with latent world models typically evaluates every candidate action sequence through a full-horizon rollout. Selective full-horizon rollouts can reduce this cost by informing estimates of the remaining candidates' evaluations. This requires modeling correlations among candidate evaluations and allocating rollouts to preserve planning updates. To this end, we propose cross-candidate inference for soft action-distribution updates in frozen world models, allocating full-horizon rollouts according to update sensitivity. Using short-horizon predicted states, evaluation trends, and remaining control inputs, the method captures correlations across candidates through a Gaussian process with a trend term. We combine the posterior covariance from this Gaussian process with the local sensitivity of action-distribution updates to derive an update-risk criterion for selecting candidates for full-horizon rollouts. Conditioning on the selected evaluations reconstructs the remaining scores, allowing all candidates to inform the soft update. On three DINO-WM tasks, our method saves 48% of prediction steps and achieves a 1.55–$2.38\times$ speedup over full evaluation with the same soft update. On the two LeWM tasks, our method reduces mean planning latency by approximately 41% relative to full evaluation.

1 INTRODUCTION

Latent world models (Ha & Schmidhuber, 2018; Hafner et al., 2019; Zhou et al., 2025; Sobal et al., 2025; Maes et al., 2026a; Assran et al., 2025) support goal-directed planning by predicting future states via action-conditioned dynamics. This planning process is often implemented by sampling-based optimization, which links changes in the action distribution to the evaluations obtained by rolling out candidate action sequences. With the help of sampling-based optimization (Chua et al., 2018; Pinneri et al., 2021), researchers have not only achieved goal-directed control in tasks such as maze navigation and object manipulation, but used learned world models to reach new goals without training task-specific policies (Zhou et al., 2025).

The acceleration of world-model planning is gaining attention due to the high computational cost of evaluating candidate action sequences (Zhou et al., 2025; Gao & Xu, 2026). To reduce planning latency and maintain control performance, existing methods have explored reducing the number or computational cost of candidate rollouts (Pinneri et al., 2021; Chun et al., 2026; Gao & Xu, 2026). Existing approaches include sample-efficient planning, where temporally correlated action sampling and elite reuse improve search efficiency (Pinneri et al., 2021), and sparse imagination, where processing fewer visual tokens reduces the cost of model prediction (Chun et al., 2026). These approaches improve candidate sampling or the computation of predicted outcomes, addressing different sources of planning cost.

In contrast to the approaches above, which improve sampling efficiency or reduce prediction costs, this paper studies planning acceleration from the perspective of cross-candidate inference. Specifically, we implement cross-candidate inference by decomposing candidate evaluation into short-horizon rollouts, selective continuation to the full horizon, and reconstruction of the remaining long-horizon evaluations. This framework can be integrated into sampling-based planning with frozen world models, without additional training of their dynamics predictors. Planning updates then use both directly computed and estimated scores, allowing all sampled candidates to contribute while only a subset is fully rolled out (Yi et al., 2024).

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.

Moreover, we also examine why more accurate estimates of candidate scores do not necessarily lead to more accurate planning updates (Elmachtoub & Grigas, 2022; Mandi et al., 2024; Bansal et al., 2023). Our explanation starts by observing that the same score error can have different effects across candidates, and that multiple errors can partly cancel or reinforce one another in planning updates. In short, a change in one candidate's score alters its weight relative to the others, changing how strongly its action sequence influences the next planning update (Yi et al., 2024). This makes score accuracy alone an *incomplete* guide for allocating a limited rollout budget. This mismatch is most serious when correcting large score errors consumes the rollout budget without meaningfully reducing uncertainty about the next planning update. Inspired by this, we allocate full-horizon rollouts to reduce uncertainty in soft action-distribution updates and use their evaluations to reconstruct scores across the candidate pool.

2 How Errors in Estimated Scores Affect Planning Updates

Planning with a latent world model searches for an action sequence whose predicted consequences bring the system closer to a specified goal. In this section, we examine how errors in estimated candidate scores affect the action-distribution update.

Latent rollouts and action-distribution updates. For a pool of N candidates indexed by $i = 0, \dots, N-1$, the i-th candidate is an action sequence $\mathbf{A}_i = (\mathbf{a}_{i,0}, \dots, \mathbf{a}_{i,H-1})$, where $\mathbf{a}_{i,t} \in \mathbb{R}^{d_a}$ is the action at prediction step t, d_a is the action dimension, and H is the planning horizon. A frozen world model predicts a latent trajectory for each candidate action sequence, and a goal-conditioned cost compares its predicted endpoint with the specified goal.

$$\widehat{\mathbf{z}}_{i,t+1} = f_\theta(\mathcal{H}_{i,t}, \mathbf{a}_{i,t}), \quad t = 0, \dots, H-1; \qquad J_i^{(H)} = \mathcal{C}(\widehat{\mathbf{z}}_{i,H}, E(o_g)), \tag{1}$$

where E and f_θ denote the frozen observation encoder and dynamics predictor, respectively, with θ denoting the predictor's fixed parameters. The history $\mathcal{H}_{i,t}$ contains the available latent representations, past actions, and any auxiliary inputs required by the predictor for candidate i at step t. Here, $\widehat{\mathbf{z}}_{i,1:H}$ denotes the latent trajectory predicted under $\mathbf{A}_i$, and o_g is the goal observation. The cost function $\mathcal{C}$ sums the mean squared errors between the predicted endpoint and $E(o_g)$ within each observation modality, using the task's fixed modality weights, giving the full-horizon cost $J_i^{(H)}$.

Candidate scores derived from **full-horizon costs** determine how strongly each action sequence contributes to the updated distribution. For each candidate, we define $s_i = -\operatorname{clip}(J_i^{(H)}, 0, J_{\max})/T$, where $T > 0$ is the temperature and $J_{\max}$ is the cost-clipping threshold. The full-evaluation reference for our method updates a diagonal Gaussian action distribution by first converting candidate scores into **normalized weights** $w_i = \exp(s_i)/\sum_{j=0}^{N-1}\exp(s_j)$ and then computing the **weighted mean and variance** of the action sequences:

$$\boldsymbol{\mu}_t^+ = \sum_{i=0}^{N-1} w_i \mathbf{a}_{i,t}, \quad \mathbf{v}_t^+ = \sum_{i=0}^{N-1} w_i \left(\mathbf{a}_{i,t} - \boldsymbol{\mu}_t^+\right)^{\odot 2}, \quad t = 0, \dots, H-1. \tag{2}$$

where $\boldsymbol{\mu}_t^+$ and $\mathbf{v}_t^+$ denote the updated mean and elementwise variance at prediction step t, respectively, the superscript $+$ indicates the updated distribution, and $\odot 2$ denotes elementwise squaring for each action coordinate.

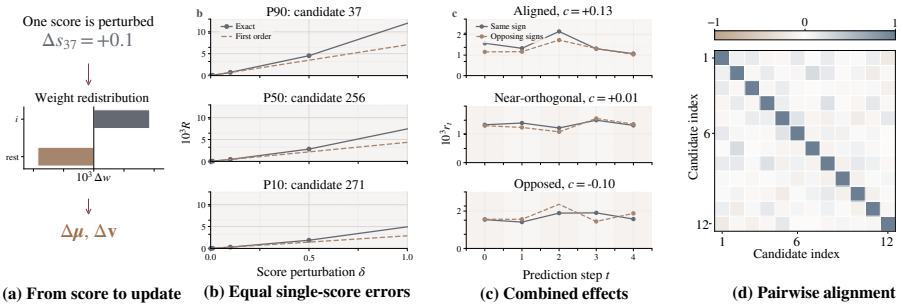
Candidate-wise influence decomposition. Errors in candidate scores propagate via the normalized weights to the updated mean and variance, *i.e.*, $\Delta\boldsymbol{\mu}_t := \boldsymbol{\mu}_t^+(\widehat{\mathbf{s}}) - \boldsymbol{\mu}_t^+(\mathbf{s})$ and $\Delta\mathbf{v}_t := \mathbf{v}_t^+(\widehat{\mathbf{s}}) - \mathbf{v}_t^+(\mathbf{s})$, where $\mathbf{s} = (s_0, \dots, s_{N-1})^\top$ collects the candidate scores defined above and $\widehat{\mathbf{s}}$ denotes its estimated counterpart, with both updates computed from the same candidate action sequences in both cases.

A first-order expansion in the **score error** $\Delta\mathbf{s} = \widehat{\mathbf{s}} - \mathbf{s}$ decomposes the changes in the updated mean and variance into candidate-wise contributions at each prediction step.

$$\Delta\boldsymbol{\mu}_t \approx \sum_{i=0}^{N-1} \frac{\partial \boldsymbol{\mu}_t^+}{\partial s_i} \Delta s_i, \quad \Delta\mathbf{v}_t \approx \sum_{i=0}^{N-1} \frac{\partial \mathbf{v}_t^+}{\partial s_i} \Delta s_i. \tag{3}$$

where Δs_i is the i-th component of $\Delta\mathbf{s}$, and the derivatives are evaluated at $\mathbf{s}$. Differentiating the update equations with respect to each candidate's score yields:

$$\frac{\partial w_j}{\partial s_i} = w_j(\delta_{ij} - w_i), \quad \frac{\partial \boldsymbol{\mu}_t^+}{\partial s_i} = \sum_{j=0}^{N-1} \mathbf{a}_{j,t} \frac{\partial w_j}{\partial s_i}, \quad \frac{\partial \mathbf{v}_t^+}{\partial s_i} = \sum_{j=0}^{N-1} \left(\mathbf{a}_{j,t} - \boldsymbol{\mu}_t^+\right)^{\odot 2} \frac{\partial w_j}{\partial s_i}. \tag{4}$$



(a) From score to update **(b) Equal single-score errors** **(c) Combined effects** **(d) Pairwise alignment**

Figure 1: Candidate-wise sensitivity and combined score-error effects on a real recorded candidate pool from DINO-WM on the Wall task. *(a)* Weight redistribution; *(b)* unequal errors from equal perturbations; *(c)* profiles for aligned, near-orthogonal, and opposed two-score errors; *(d)* derivative alignment. All curves are exact recomputations from the same recorded pool rather than fitted trends.

Where δ_{ij} is the Kronecker delta, equals 1 when the score being differentiated and the weight being affected belong to the same candidate, and 0 otherwise; the normalization couples all candidates.

An error in one candidate's score changes not only its own normalized weight but also the weights assigned to every other candidate. The magnitude of a score error alone therefore does not determine its influence on the update, since that influence also depends on the candidate's normalized weight and its action sequence relative to the rest of the pool. The overall update error therefore depends not only on the influence of each score error, but also on how the resulting contributions align.

Combined influence and a demonstration on Wall. We illustrate these effects with a navigation example using DINO-WM on the Wall task (Zhou et al., 2024). We use 900 recorded candidate pools (each containing the $N{=}300$ candidates from one planning update) from 50 Wall tasks, with horizon $H{=}5$. We denote the full-horizon rollout budget by B, the number of candidates selected for full-horizon rollouts from each pool. Within each pool, we probe the local response by evaluating a nearby score value for each candidate in turn at a common offset, and measure the resulting changes in the updated mean and variance. To put both moments on the scale used for rollout allocation, we first evaluate each candidate to a prefix length $h{<}H$. The prefix utilities $u_i^{(h)} = \exp[-\mathrm{clip}(J_i^{(h)}, 0, J_{\max})/T]$, where $J_i^{(h)} = \mathcal{C}(\widehat{\mathbf{z}}_{i,h}, E(o_g))$, give normalized prefix weights $p_i = u_i^{(h)} / \sum_j u_j^{(h)}$. Using $w_i = p_i$ in Eq. (2) gives the provisional mean $\widetilde{\boldsymbol{\mu}}_t$ and variance $\widetilde{\mathbf{v}}_t$. We measure the resulting update error by $R \geq 0$, defined through

$$R^2 = \frac{1}{d} \sum_{t=0}^{H-1} \left(\left\| \frac{\Delta\boldsymbol{\mu}_t}{\sqrt{\widetilde{\mathbf{v}}_t}} \right\|_2^2 + \frac{1}{2} \left\| \frac{\Delta\mathbf{v}_t}{\widetilde{\mathbf{v}}_t} \right\|_2^2 \right), \tag{5}$$

where $d{=}Hd_a$ is the total number of action coordinates across the horizon. Division and square roots are elementwise. The prefix length is $h{=}2$ in these experiments, and the provisional variance is held fixed within each pool via all perturbations and allocation comparisons.

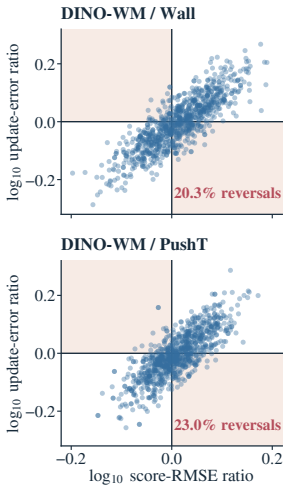


Figure 2: Ranking reversals between score and update accuracy at $B{=}60$.

Figure 1(a) shows that changing one score redistributes normalized weight across the candidate pool, while (b) shows that the same score perturbation produces different update errors at different sensitivity levels. Figure 1(c) and (d) show that the joint effect of multiple errors depends on the alignment of their update derivatives: errors can reinforce or oppose one another. **Together, these panels show why improving individual score estimates need not improve the combined planning update under a fixed rollout budget.**

Score accuracy and rollout allocation. To test this beyond the controlled perturbations, we compare total-score-variance allocation with update-aware allocation based on planning-update sensitivity at a common budget of $B{=}60$, using the same reconstruction model and posterior mean; the evaluation protocol is summarized in the Appendix. After removing the common score offset, Figure 2

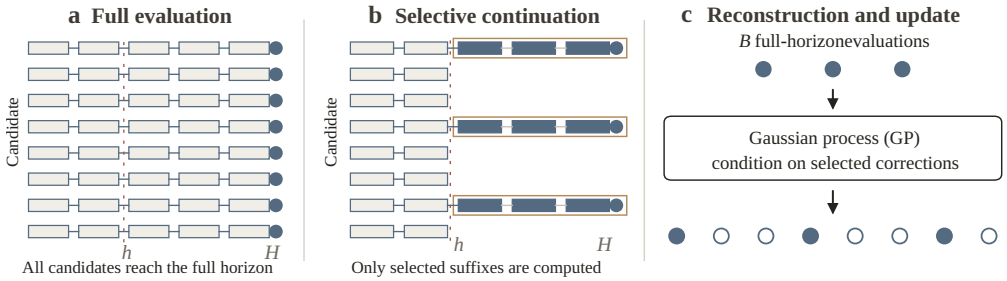


Figure 3: Cross-candidate inference with selective continuation. *(a)* Full evaluation rolls every candidate to horizon H; *(b)* All candidates share a prefix of length h, but only the selected B candidates continue to H; *(c)* Their evaluations condition the GP to reconstruct the remaining scores.

plots the log ratios of centered score root-mean-square error (RMSE), computed between the centered estimated and full-horizon score vectors, and update error for all 900 Wall and 900 PushT (Chi et al., 2024; Zhou et al., 2025) pools. Opposite-sign coordinates mark ranking reversals: the rule with lower centered score RMSE has higher update error in 20.3% of Wall pools and 23.0% of PushT pools, with both directions represented. **These realized-update reversals motivate allocating rollouts according to how correlated candidate scores affect the joint update, rather than score accuracy alone.** Additional protocol details, task-wise intervals, and the oracle-correction diagnostic are provided in the Appendix. The next section combines this sensitivity analysis with cross-candidate reconstruction to formulate a rollout-allocation criterion guided by the sensitivity of planning updates to score errors.

3 From Score Errors to Cross-Candidate Inference

Overview. For the soft update in Eq. (2), the analysis in Section 2 shows that candidate score errors are coupled through normalized weights and the resulting weighted moments. This coupling motivates an update-aware objective for allocating a limited full-horizon rollout budget: select candidates whose full-horizon evaluations are expected to reduce uncertainty in the next action-distribution update. Specifically, we value a full-horizon rollout by propagating uncertainty in the unobserved long-horizon score corrections through the first-order score-to-update relation in Eq. (3), with its standardized local risk given by Eq. (6). Our method therefore uses short-horizon latent states and remaining control inputs to model correlations in long-horizon score corrections through a Gaussian process (GP). Combining the GP posterior uncertainty with this local update risk, we select candidates whose full-horizon evaluations are expected to reduce the posterior risk of the next action-distribution update. The selected full-horizon evaluations then condition the same GP to reconstruct the remaining score corrections, and all candidates enter the soft update with directly evaluated or reconstructed scores. Figure 3 summarizes how selective continuation and cross-candidate reconstruction retain all candidates in the planning update.

Eq. (3) states that a candidate score error $\Delta\mathbf{s}$ changes the weighted mean $\boldsymbol{\mu}_t^+$ and weighted variance $\mathbf{v}_t^+$ through the candidate-wise sensitivities $\partial\boldsymbol{\mu}_t^+/\partial s_i$ and $\partial\mathbf{v}_t^+/\partial s_i$. These sensitivities are obtained from the normalized-weight derivative $\partial w_j/\partial s_i$ in Eq. (4), so the error Δs_i of candidate score s_i contributes a specific direction to the joint moment change $\left(\Delta\boldsymbol{\mu}_t, \Delta\mathbf{v}_t\right)$. For rollout allocation, we evaluate these derivatives at the available prefix scores $\log u_i^{(h)}$ and hold this operating point fixed while selecting the batch. Using the provisional variance $\widetilde{\mathbf{v}}_t$ from Eq. (5), the **local update risk** is

$$R^2_{\mathrm{lin}}(\Delta\mathbf{s}) := \frac{1}{d}\sum_{t=0}^{H-1}\left[\left\|\frac{\sum_{i=0}^{N-1}\frac{\partial\boldsymbol{\mu}_t^+}{\partial s_i}\Delta s_i}{\sqrt{\widetilde{\mathbf{v}}_t}}\right\|_2^2 + \frac{1}{2}\left\|\frac{\sum_{i=0}^{N-1}\frac{\partial\mathbf{v}_t^+}{\partial s_i}\Delta s_i}{\widetilde{\mathbf{v}}_t}\right\|_2^2\right]. \tag{6}$$

The posterior expectation of this local risk guides rollout allocation; the final update evaluates Eq. (2) using the reconstructed scores.

3.1 Modeling Long-Horizon Score Corrections

We model long-horizon score corrections jointly so that each selected rollout can reduce uncertainty across the candidate pool. Before selecting the full-horizon rollouts, we roll out every candidate to a

common prefix length $h<H$ and construct a **state–control feature** ϕ_i from the resulting latent state $\widehat{\mathbf{z}}_{i,h}$, the goal representation $E(o_g)$, and the remaining actions $(\mathbf{a}_{i,h},\dots,\mathbf{a}_{i,H-1})$. The shared prefix yields an unnormalized **prefix utility** $u_i^{(h)}$ for every candidate, whereas the **full-horizon utility** $u_i^{(H)}$ is observed only for candidates selected for continuation. At each evaluated horizon L, we use $u_i^{(L)}=\exp[-\operatorname{clip}(J_i^{(L)},0,J_{\max})/T]$, with $J_i^{(L)}=\mathcal{C}(\widehat{\mathbf{z}}_{i,L},E(o_g))$. Thus $\log u_i^{(H)}=s_i$. The reference candidate 0 follows the current sampling mean and is always included in the full-horizon budget of B candidates. To compare corrections across candidates, we define the **long-horizon score correction** y_i as the relative log-utility change from $u_i^{(h)}$ to $u_i^{(H)}$, anchored at candidate 0:

$$y_i := \log \frac{u_i^{(H)} u_0^{(h)}}{u_i^{(h)} u_0^{(H)}}, \qquad i=0,\dots,N-1. \tag{7}$$

Subtracting the reference correction removes a common score shift, which leaves the normalized weights unchanged. For an earlier prefix $h_1<h$, the available utilities also define the **prefix trend** $\tau_i := \log(u_i^{(h)}/u_i^{(h_1)}) - \log(u_0^{(h)}/u_0^{(h_1)})$, the reference-relative log-utility change over the shared prefix. For posterior inference, we use τ_i as the mean-function input and use ϕ_i to couple the residuals g_i across candidates:

$$y_i = \beta\,\tau_i + g_i, \qquad \mathbf{g} := (g_0,\dots,g_{N-1})^\top \sim \mathcal{N}(\mathbf{0},K). \tag{8}$$

The trend coefficient β is estimated within each candidate pool from the selected full-horizon evaluations. We construct the **residual covariance matrix** $K=P_0k^{(0)}P_0^\top$ using the RBF kernel $k_{ij}^{(0)} := \sigma_f^2\exp[-\|\phi_i-\phi_j\|_2^2/(2d_\phi\ell_\phi^2)]$ and the reference-centering matrix $P_0 := \mathbf{I}_N-\mathbf{1}\mathbf{e}_0^\top$. Candidates with similar prefix states and remaining controls therefore receive correlated residual corrections. Here $\mathbf{I}_N$ is the $N\times N$ identity matrix, $\mathbf{1}$ is the N-dimensional all-ones vector, and $\mathbf{e}_0$ is the basis vector selecting candidate 0. Thus, P_0 maps each candidate-indexed quantity to its difference from the reference candidate, giving $K_{0:}=K_{:0}=0$, where a colon in a matrix subscript selects all indices along that dimension. We denote the feature dimension by $d_\phi=\dim(\phi_i)$ and the base covariance matrix by $k^{(0)}=[k_{ij}^{(0)}]_{i,j=0}^{N-1}$, with kernel amplitude $\sigma_f>0$ and length scale $\ell_\phi>0$.

For each **observation modality** b (an input type such as images or robot joint states), we pool the predicted latent state $\widehat{\mathbf{z}}_{i,h}$ and the goal representation $E(o_g)$ by averaging temporal or patch dimensions when present, obtaining vectors $\bar{\mathbf{z}}_{i,h}^{(b)}$ and $\bar{\mathbf{z}}_g^{(b)}$, respectively. For candidate i and observation modality b, the **displacement** $\mathbf{x}_{i,b}:=\bar{\mathbf{z}}_{i,h}^{(b)}-\bar{\mathbf{z}}_g^{(b)}$ describes where the predicted state lies relative to the goal in latent space. For each observation modality b, let $r_{i,b} := \max\{\|\mathbf{x}_{i,b}\|_2,\varepsilon\}$. We retain the displacement direction and standardize its log length across the pool to obtain

$$\mathbf{x}_{i,b}^{\mathrm{goal}} := \frac{1}{\sqrt{2}}\left[\frac{\mathbf{x}_{i,b}}{r_{i,b}}\,;\,\frac{\log r_{i,b}-\operatorname{mean}_j(\log r_{j,b})}{\max\{\operatorname{std}_j(\log r_{j,b}),\varepsilon\}}\right]. \tag{9}$$

The floor $\varepsilon>0$ keeps logarithms and divisions defined, while $[\,;\,]$ and concat denote vector concatenation, and mean_j and std_j compute the mean and population standard deviation across the N candidates separately for each observation modality b. Let $\mathcal{B}$ denote the set of observation modalities. We concatenate their features $\mathbf{x}_{i,b}^{\mathrm{goal}}$ and normalize by $\sqrt{|\mathcal{B}|}$ to obtain $\mathbf{x}_i^{\mathrm{goal}}:=|\mathcal{B}|^{-1/2}\,\mathrm{concat}_{b\in\mathcal{B}}(\mathbf{x}_{i,b}^{\mathrm{goal}})$. To construct the control component of ϕ_i, we summarize the remaining actions by their mean $\bar{\mathbf{a}}_i:=(H-h)^{-1}\sum_{t=h}^{H-1}\mathbf{a}_{i,t}$ and empirical covariance $S_i := (H-h)^{-1}\sum_{t=h}^{H-1}(\mathbf{a}_{i,t}-\bar{\mathbf{a}}_i)(\mathbf{a}_{i,t}-\bar{\mathbf{a}}_i)^\top$. The **control feature** $\mathbf{c}_i$ concatenates the normalized direction and standardized log length of $\bar{\mathbf{a}}_i$, computed with the same numerical floors as in Eq. (9), with the normalized spectrum $\boldsymbol{\lambda}(S_i)/\max\{\mathrm{tr}(S_i),\varepsilon\}$. Here tr denotes the matrix trace, and $\boldsymbol{\lambda}(S_i)$ lists the eigenvalues from largest to smallest, describing how the remaining controls vary across principal directions. To put the state and control components on a common scale, let $r_x^2:=N^{-1}\sum_j\|\mathbf{x}_j^{\mathrm{goal}}\|_2^2$ and $r_c^2:=N^{-1}\sum_j\|\mathbf{c}_j\|_2^2$. The resulting feature is $\phi_i:=2^{-1/2}[\mathbf{x}_i^{\mathrm{goal}}/\max\{r_x,\varepsilon\}\,;\,\mathbf{c}_i/\max\{r_c,\varepsilon\}]$. All components are available after the shared prefix, so constructing these correlations requires no additional world-model predictions.

3.2 Allocating Rollouts by Planning-Update Sensitivity

The scores available during rollout allocation are the prefix log utilities, since the long-horizon score corrections have not yet been observed. The prefix weights p_i and provisional moments defined in

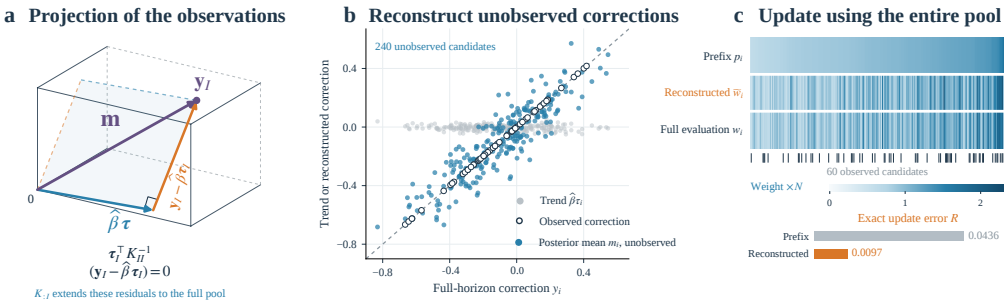


Figure 4: Cross-candidate reconstruction from selected evaluations. *(a)* Schematic trend–residual decomposition: on the observed indices I, the posterior mean satisfies $\mathbf{m}_I{=}\mathbf{y}_I$, and the residual is orthogonal to the trend in the K_{II}^{-1} metric. Cross-covariance $K_{:I}$ extends the residual correction to the full pool; *(b)* Posterior means reconstruct the unobserved corrections, compared with the fitted trend and full-horizon reference values; *(c)* Prefix, reconstructed, and full-evaluation weights share the same prefix-weight ordering. The update error R decreases from 0.0436 to 0.0097 in this pool. (b) and (c) use one recorded Wall pool with $N{=}300$ and $B{=}60$.

Section 2 therefore provide all quantities needed to evaluate Eq. (6). Simplifying the derivatives in Eq. (4) at these scores and stacking their standardized mean and variance components gives the **candidate-wise update sensitivity**:

$$\mathbf{G}_i \coloneqq \frac{p_i}{\sqrt{d}}\,\mathrm{concat}_{t=0}^{H-1}\left[\frac{\mathbf{a}_{i,t}-\widetilde{\boldsymbol{\mu}}_t}{\sqrt{\widetilde{\mathbf{v}}_t}}\,;\,\frac{(\mathbf{a}_{i,t}-\widetilde{\boldsymbol{\mu}}_t)^{\odot 2}-\widetilde{\mathbf{v}}_t}{\sqrt{2}\,\widetilde{\mathbf{v}}_t}\right]. \tag{10}$$

Stacking these sensitivities gives $G{\coloneqq}[\mathbf{G}_0,\dots,\mathbf{G}_{N-1}]^\top \in \mathbb{R}^{N\times 2d}$, which maps a score-error vector to the standardized local moment change through $G^\top\Delta\mathbf{s}$. Thus $R^2_{\mathrm{lin}}(\Delta\mathbf{s}){=}\|G^\top\Delta\mathbf{s}\|_2^2$. The identity $G^\top\mathbf{1}{=}0$ also shows why anchoring the corrections at candidate 0 preserves local effect on update.

Posterior update risk. Let $\mathcal{S}$ be a set of candidates selected for continuation, including candidate 0, and write $I{=}\mathcal{S}\backslash\{0\}$. Subscript I restricts a vector to the selected non-reference candidates. We collect the corrections in $\mathbf{y}{=}(y_0,\dots,y_{N-1})^\top$ and denote their posterior mean and covariance given $\mathbf{y}_I$ by $\mathbf{m}\in\mathbb{R}^N$ and $C{=}C(\mathcal{S}){\in}\mathbb{R}^{N\times N}$. Propagating the posterior-mean error $\mathbf{m}{-}\mathbf{y}$ through G gives the expected local update risk. The common reference shift cancels because $G^\top\mathbf{1}=0$:

$$\mathcal{R}(C) \coloneqq \mathbb{E}\big[R^2_{\mathrm{lin}}(\mathbf{m}-\mathbf{y})\,\big|\,\mathbf{y}_I\big] \qquad = \mathbb{E}\Big[\big\|G^\top(\mathbf{m}-\mathbf{y})\big\|_2^2\,\Big|\,\mathbf{y}_I\Big] = \mathrm{tr}\big(G^\top C G\big)\,. \tag{11}$$

Equivalently, $\mathcal{R}(C)=\sum_{i,j}C_{ij}\,\mathbf{G}_i{}^\top\mathbf{G}_j$. The off-diagonal terms combine score correlations with the alignment of candidate sensitivities, retaining the reinforcement and cancellation identified in Section 2. This gives a computable objective for allocating rollouts under the local model.

Because β in Eq. (8) is estimated from the selected candidates, the posterior must account for uncertainty in the trend as well as in the residuals. We integrate β under a flat prior and write $\boldsymbol{\tau}{=}(\tau_0,\dots,\tau_{N-1})^\top$. Gaussian conditioning gives the covariance associated with a proposed set $\mathcal{S}$:

$$C(\mathcal{S}) = \underbrace{K - K_{:I}K_{II}^{-1}K_{I:}}_{\text{residual uncertainty}} + \underbrace{\frac{\mathbf{r}\mathbf{r}^\top}{\boldsymbol{\tau}_I{}^\top K_{II}^{-1}\boldsymbol{\tau}_I}}_{\text{trend uncertainty}}. \tag{12}$$

Here K_{II} is the covariance among the selected non-reference candidates, $K_{:I}$ is their cross-covariance with the full pool, and $\mathbf{r}\coloneqq\boldsymbol{\tau}-K_{:I}K_{II}^{-1}\boldsymbol{\tau}_I\in\mathbb{R}^N$ has one entry per candidate in the full pool. The expression assumes invertible K_{II} and positive trend information $\boldsymbol{\tau}_I{}^\top K_{II}^{-1}\boldsymbol{\tau}_I{>}0$. Its dependence on the selected indices, features, and prefix trends, but not on the unknown values $\mathbf{y}_I$, allows us to evaluate rollout sets before computing their full-horizon scores.

Selecting a batch by risk reduction. We first include candidate 0 and choose the first non-reference candidate to minimize $\mathcal{R}(C(\{0,j\}))$ over candidates with $K_{jj}{>}0$ and $\tau_j{\neq}0$. This makes the trend identifiable and yields a finite covariance from which to select the remaining candidates. If the trend vanishes throughout the pool, we set the trend-coefficient estimate $\widehat{\beta}{=}0$, omit the trend-uncertainty term, and start selection from $C{=}K$. For any unselected candidate j with $C_{jj}{>}0$, let

$C^+ = C(\mathcal{S} \cup \{j\})$ denote the posterior covariance after observing its correction. Noiseless conditioning gives the rank-one update $C^+ = C - C_{:j}C_{j:}/C_{jj}$ and an analytic decrease in update risk:

$$\mathcal{R}(C) - \mathcal{R}(C^+) = \frac{\|G^\top C_{:j}\|_2^2}{C_{jj}}, \qquad j^\star = \underset{j \notin \mathcal{S},\, C_{jj}>0}{\arg\max} \frac{\|G^\top C_{:j}\|_2^2}{C_{jj}}. \tag{13}$$

We append $j^\star$ to $\mathcal{S}$, apply the covariance update, and repeat until $|\mathcal{S}|=B$. The covariance column $C_{:j}$ describes which candidates can be informed by evaluating j, while $G^\top$ measures how that information affects their joint update. For example, a candidate with small individual sensitivity can still be valuable if its evaluation informs other candidates with a strong combined influence. **Our rule therefore selects full-horizon rollouts for the reduction they induce in planning-update uncertainty across the entire candidate pool.** The greedy covariance updates account for information covered by earlier selections, while all B continuations can be executed after the batch is selected.

3.3 Reconstructing Scores for the Full Candidate Pool

The selected candidates now continue from their cached histories $\mathcal{H}_{i,h}$ to horizon H, yielding the observed corrections $\mathbf{y}_I$ through Eq. (7). Using these observations in the same GP, generalized least squares gives $\widehat{\beta} = (\boldsymbol{\tau}_I{}^\top K_{II}^{-1}\mathbf{y}_I)/(\boldsymbol{\tau}_I{}^\top K_{II}^{-1}\boldsymbol{\tau}_I)$. We reconstruct all candidates' corrections:

$$\mathbf{m} = \widehat{\beta}\,\boldsymbol{\tau} + K_{:I}K_{II}^{-1}(\mathbf{y}_I - \widehat{\beta}\,\boldsymbol{\tau}_I). \tag{14}$$

The first term calibrates the prefix trend to the full-horizon observations, and the second propagates their residual corrections across candidates. The covariance is the same $C(\mathcal{S})$ used for allocation in Eq. (12). For every observed candidate $i \in \mathcal{S}$, conditioning gives $m_i = y_i$ and $C_{ii} = 0$, including $m_0 = 0$ at the reference.

To incorporate the remaining uncertainty into planning, we use the Gaussian certainty equivalent $m_i - \frac{1}{2}C_{ii}$ with a fixed unit risk coefficient. This equals $-\log \mathbb{E}[\exp(-y_i)|\mathbf{y}_I]$ under the posterior and penalizes uncertain reconstructions while leaving observed corrections unchanged. The resulting estimated scores and their normalized weights are

$$\widehat{s}_i = \log u_i^{(h)} + m_i - \tfrac{1}{2}C_{ii}, \qquad \widetilde{w}_i = \frac{\exp(\widehat{s}_i)}{\sum_j \exp(\widehat{s}_j)}. \tag{15}$$

The variance penalty specifies the risk preference of the final update, and the posterior-mean comparison in Section 2 holds the reconstruction rule fixed to isolate rollout allocation. For selected candidates, $\widehat{s}_i = s_i - (\log u_0^{(H)} - \log u_0^{(h)})$, so their observed relative scores are preserved exactly. Substituting $\widetilde{w}_i$ for w_i in Eq. (2) then gives $\boldsymbol{\mu}_t^+(\widehat{\mathbf{s}})$ and $\mathbf{v}_t^+(\widehat{\mathbf{s}})$ from all N candidates. **Each selected rollout thus serves both as a direct evaluation and as evidence for reconstructing the remaining candidates' contributions to the planning update.** Figure 4 illustrates this decomposition and its effect on reconstructed weights and update error in a recorded Wall pool.

Planning and computational cost. After several action-distribution updates, we execute the action prefix specified by each model's control protocol and replan from the resulting observation. Reusing each selected candidate's prefix requires $Nh + B(H-h)$ candidate prediction steps per update, compared with NH for full evaluation of the pool. The fraction of prediction steps saved is $(1-B/N)(1-h/H)$; the encoder and dynamics predictor remain frozen throughout. Planning latency also includes feature construction, kernel inference, and batching world-model predictions.

4 Experiments

Experimental setup. We evaluate frozen DINO-WM (Zhou et al., 2024) on PointMaze (Fu et al., 2020), Wall (Zhou et al., 2024), and PushT (Chi et al., 2025), and LeWM (Maes et al., 2026b) on PushT and Cube (Park et al., 2025), using 50 fixed start–goal tasks and three planner seeds per setting (150 episodes). We compare against Full-soft (evaluating all candidates), Reduced-N (fully evaluating fewer candidates to match our prediction work), Uniform (random selection for full-horizon rollouts), and total-score-variance allocation (greedily reducing total posterior score variance). All methods retain soft updates and share the checkpoint and execution budget within each setting, and the latter two also share our candidate pool, rollout budget, and reconstruction model. The default is N=300, B=60, H=5, h=2, with ten optimization rounds for DINO-WM and 30 for LeWM. Primary metrics are task success and measured planning latency. 95% intervals use 10,000 task-cluster bootstrap resamples. Experiments use 11 NVIDIA H200 GPUs. Appendix C specifies the task manifests and execution protocols.

Table 1: Success rate (SR, %) and mean latency (s). Timing covers observation to action for DINO-WM and the solver for LeWM. Details appear in Appendices C–D.

	DINO-WM						LeWM			
	PointMaze		Wall		PushT		PushT		Cube	
Method	SR ↑	Time ↓	SR ↑	Time ↓	SR ↑	Time ↓	SR ↑	Time ↓	SR ↑	Time ↓
Ours	61.3	6.092	58.0	3.266	**27.3**	6.060	**36.7**	2.036	**53.3**	**2.077**
Full-soft	**61.7**	9.752	**58.7**	5.069	26.0	14.418	36.0	3.493	**53.3**	3.490
Reduced-N	52.7	7.581	54.7	**2.723**	24.1	7.568	32.0	**1.893**	48.0	2.896
Uniform	54.0	**6.080**	55.3	3.265	22.0	**6.054**	25.0	2.070	49.2	2.141
Total-score variance	52.3	6.099	56.0	3.286	25.3	6.072	31.3	2.148	50.3	2.095

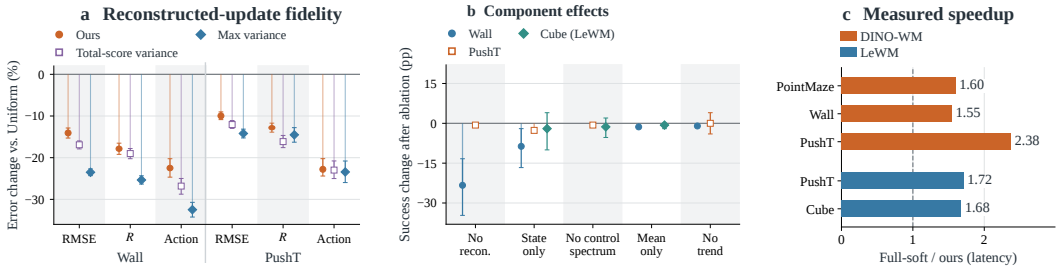


Figure 5: Complementary tests of cross-candidate inference. *(a)* Error change relative to Uniform: correction RMSE, update error R, and first-action error, lower is better; *(b)* Success change after component ablations; *(c)* Within-model speedup over Full-soft.

Planning quality and efficiency. Cross-candidate inference saves 48% of candidate prediction steps and gives a measured 1.55–$2.38\times$ **speedup over Full-soft on DINO-WM** (Figure 5(c)). Success rates remain close to Full-soft, changing by -0.4, -0.7, and $+1.3$ percentage points (pp) on PointMaze, Wall, and PushT, respectively (Table 1). On LeWM, the speedup is 1.68–$1.72\times$, with success changes of $+0.7$ pp on PushT and 0.0 pp on Cube. The reported latencies include feature construction, rollout selection, and GP conditioning, so the speedups account for the additional inference cost. At matched prediction work, our method improves success over Reduced-N by 3.2–8.6 pp across the five settings; Reduced-N is faster on Wall and LeWM PushT. With the same prediction budget and nearly identical latency, our method also exceeds Uniform by 7.3, 2.7, and 5.3 pp on the three DINO-WM tasks. These gains demonstrate the value of allocating full-horizon evaluations according to their effect on the planning update.

From score reconstruction to planning updates. We evaluate our update-aware allocation rule (Eq. (13)) using 900 recorded candidate pools for each of Wall and PushT, holding candidate actions, reference evaluations, and reconstruction fixed at $B{=}60$. The pools span two trajectory sources and three optimization stages, with uncertainty aggregated over the 50 tasks. Figure 5(a) shows that our allocation rule reduces R relative to Uniform by approximately 18% on Wall and 13% on PushT. Our method also reduces first-action error by approximately 23% relative to Uniform on both tasks. First-action error uses the weighted mean before clipping, directly testing whether reconstruction preserves the command proposed by the full-evaluation update. Among the methods shown, maximum-variance allocation (Max variance) achieves the lowest R on Wall, whereas total-score-variance allocation achieves the lowest R on PushT. On PushT, maximum-variance allocation has lower correction RMSE than total-score-variance allocation, reversing their ranking by R. **The ranking by score accuracy can therefore differ from the ranking by update accuracy**, consistent with the sensitivity analysis in Section 2. The shared pools rule out changes in sampled actions as an explanation for these differences. Complete reconstruction results are reported in Appendix D.4.

Propagating observed corrections across candidates improves control at a fixed rollout budget. Removing reconstruction keeps the selected full-horizon evaluations but leaves unselected candidates at their prefix scores. On Wall, this lowers success by 23.33 pp (95% interval $[-34.67, -13.33]$, and Figure 5(b)) and increases terminal distance from 9.08 to 12.96. On PushT, removing reconstruction also lowers success, by approximately 0.7 pp. These gains come from using the selected evaluations to correct the unobserved candidates, since the directly evaluated candidates remain available in both variants. Removing the remaining-control features lowers success by 8.67 pp on Wall (95% interval $[-16.67, -2.00]$) and approximately 2.7 pp on PushT. This supports using both prefix endpoints and remaining controls when transferring corrections: similar prefix

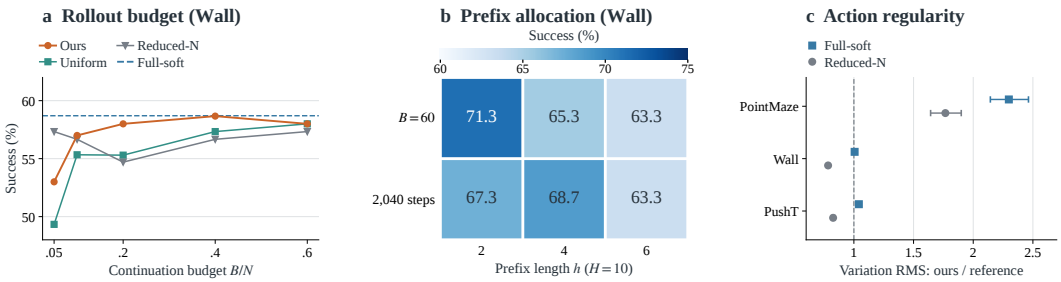


Figure 6: Prediction budgets and action regularity. *(a)* Wall success versus continuation budget B/N; the dashed line denotes Full-soft; *(b)* Wall success (%) at H=10 with fixed B=60 or fixed work of 2,040 candidate prediction steps per optimization round; *(c)* Action-variation RMS on DINO-WM tasks, expressed as ours/reference; values below one indicate smoother commands. Error bars in (c) show paired 95% task-bootstrap confidence intervals. Additional results are reported in Appendices D.6 and C.2.

endpoints can lead to different continuations. Removing the prefix trend, control spectrum, or uncertainty penalty produces smaller success changes, ranging from 0.0 to 1.3 pp across the evaluated settings. Complete ablation results are reported in Appendix D.5.

Allocating work across candidates and time. Increasing the continuation budget yields diminishing gains on Wall. Continuing only B=30 of the N=300 candidates gives 57.0% success using 46% of Full-soft's candidate prediction steps (Figure 6(a)). The remaining 90% of candidates still contribute to the update through reconstructed scores. Doubling B to 60 raises success to 58.0% with 52% of the prediction work, compared with Full-soft's 58.7% (Table 1). Our allocation rule exceeds Uniform by 1.7–3.7 pp at B=15, 30, 60, while the two tie at B=180. The benefit of selecting informative candidates is thus concentrated at small budgets. The best tested prefix length depends on the work constraint. On Wall at H=10, fixing B=60 favors h=2, which achieves 71.33% success (Figure 6(b)). With work fixed at 2,040 candidate prediction steps per optimization round, h=4 instead gives the highest success, 68.67%, with B=140. Under fixed work, a longer prefix provides more information for every candidate but permits fewer full-horizon evaluations. Changing h also changes the state–control features used to transfer corrections across candidates. The result at h=4 suggests that richer prefix information can compensate for fewer full-horizon evaluations. Prefix length and continuation budget therefore need to be chosen together. Complete sweeps are reported in Appendix D.6.

Action regularity and execution length. At matched prediction work, our method produces smoother commands than Reduced-N on DINO-WM Wall and PushT. Action-variation RMS decreases by 21.6% and 17.4%, respectively (Figure 6(c)). The paired ratios are $0.784\ [0.776, 0.792]$ and $0.826\ [0.814, 0.838]$, with 95% confidence intervals in brackets. Action variation on these tasks remains close to Full-soft's, with ratios of 1.006 and 1.042 despite using 48% fewer candidate prediction steps. This behavior is consistent with retaining contributions from the full candidate pool in the soft update. The effect on action regularity depends on the task and world model. We compare execution lengths with Reduced-N on task–seed pairs where both methods succeed. The paired 95% intervals for the differences include zero on all three DINO-WM tasks. Appendix C.2 reports the full control metrics and execution-length comparisons across tasks and world models.

5 Conclusion

Cross-candidate inference allows a full candidate pool to inform planning while only a subset receives full-horizon evaluation. To guide these evaluations, we first analyze how score errors jointly affect soft action-distribution updates. This analysis explains why score accuracy alone is insufficient for allocating a limited rollout budget. We then combine a Gaussian-process model of long-horizon score corrections with planning-update sensitivity to select informative rollouts and reconstruct the remaining scores. Experiments on DINO-WM and LeWM demonstrate measured speedups of 1.55–2.38× over Full-soft, with success rates within 1.3 percentage points of the reference across the five evaluated settings. The method requires no additional training and keeps the encoder and dynamics predictor frozen. These results establish cross-candidate inference as a practical approach to accelerating planning with latent world models.

REFERENCES

Mahmoud 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. *arXiv preprint arXiv:2506.09985*, 2025.

Dishank Bansal, Ricky TQ Chen, Mustafa Mukadam, and Brandon Amos. Taskmet: Task-driven metric learning for model learning. *Advances in Neural Information Processing Systems*, 36: 46505–46519, 2023.

Yutian Cheng, Xiaojian Ma, Xianhao Wang, Min Yang, Rongpeng Su, Hangxin Liu, Xi Chen, Shuai Li, and Qing Li. AdaReP: Adaptive re-planning under model mismatch for neural world-model predictive control. *arXiv preprint arXiv:2606.23079*, 2026. URL https://arxiv.org/abs/2606.23079.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 43(3):384–408, 2024. doi: 10.1177/02783649241273668. URL https://arxiv.org/abs/2303.04137.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Junha Chun, Youngjoon Jeong, and Taesup Kim. Sparse imagination for efficient visual world model planning. In *International Conference on Learning Representations*, 2026.

Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management science*, 68(1): 9–26, 2022.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

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

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2555–2565. PMLR, 2019.

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

Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Oxh5CstDJU.

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

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*, 2026b.

Jayanta Mandi, James Kotary, Senne Berden, Maxime Mulamba, Victor Bucarey, Tias Guns, and Ferdinando Fioretto. Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *Journal of Artificial Intelligence Research*, 80:1623–1701, 2024.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. In *International Conference on Learning Representations*, volume 2025, pp. 94937–94982, 2025.

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 *Proceedings of the 2020 Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pp. 1049–1065. PMLR, 2021.

Uladzislau Sobal, Wancong Zhang, Kyunghyun Cho, Randall Balestriero, Tim G. J. Rudner, and Yann LeCun. Learning from reward-free offline data: A case for planning with latent dynamics models. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Zeji Yi, Chaoyi Pan, Guanqi He, Guannan Qu, and Guanya Shi. Covo-mpc: Theoretical analysis of sampling-based mpc and optimal covariance design. In *6th Annual Learning for Dynamics & Control Conference*, pp. 1122–1135. PMLR, 2024.

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.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 79115–79135. PMLR, 2025.

A Related Work

A.1 World Models for Prediction and Control

World models support control by predicting the consequences of actions in a learned representation. Early work learns compact visual dynamics for controller optimization (Ha & Schmidhuber, 2018), while PlaNet performs online planning through a latent dynamics model learned from pixels (Hafner et al., 2019). Dreamer learns policies and value functions from imagined latent trajectories (Hafner et al., 2020). TD-MPC2 combines latent trajectory optimization with learned reward and terminal-value estimates, scaling this approach across continuous-control tasks (Hansen et al., 2024). These methods illustrate different ways to connect predictive representations with behavior: a world model can support direct trajectory optimization, policy learning through imagination, or planning guided by learned values. A related line of work learns predictive representations from reward-free data and specifies goals at planning time. DINO-WM predicts pretrained DINOv2 patch features and optimizes action sequences toward image-goal features (Zhou et al., 2025). PLDM studies planning with latent dynamics learned from reward-free offline trajectories, including generalization to unseen navigation layouts (Sobal et al., 2025). LeWM jointly trains its image encoder and latent predictor using a prediction objective and distributional regularization (Maes et al., 2026a). V-JEPA 2 combines large-scale video pretraining with action-conditioned post-training on robot trajectories to support planning toward visual goals (Assran et al., 2025). Across these designs, deploying a frozen model still requires substantial computation when many candidate sequences are repeatedly evaluated. **Our work addresses this planning-time cost after the world model has been trained**, preserving its representation, parameters, and goal-evaluation function.

The relationship between prediction and decision quality also matters for how computation is allocated. Decision-focused learning evaluates predictions through their downstream optimization consequences (Elmachtoub & Grigas, 2022, Mandi et al., 2024), and TaskMet learns a task-dependent metric for model training (Bansal et al., 2023). Our analysis uses planning-update sensitivity to identify consequential score uncertainty during inference. The resulting allocation rule operates on a fixed world model and the current candidate pool, connecting this perspective to the cost of obtaining evaluations.

A.2 Efficient Planning with World Models

Planning efficiency can be improved at several stages of the control pipeline. PETS combines learned probabilistic dynamics with sampling-based action optimization (Chua et al., 2018). iCEM improves sampling efficiency through temporally correlated action sequences, elite reuse, and a decreasing population (Pinneri et al., 2021), while CoVO-MPC studies covariance design for sampling-based control (Yi et al., 2024). These methods improve how candidate sequences are proposed and searched. Our method takes a candidate pool as given and reduces the work needed to obtain the scores used by its soft action-distribution update. Recent visual world models also accelerate the computation of individual candidate predictions. **Sparse Imagination** (Chun et al., 2026) performs rollouts with subsets of visual patch tokens and trains its predictor with randomized grouped attention to support sparse inputs. Its published approach therefore couples inference-time token reduction with a modified world-model training procedure. **Fast-LeWM** (Gao & Xu, 2026) introduces an action-prefix encoder and a predictor trained with supervision at multiple prefix lengths. It predicts future latents directly from action prefixes, reducing the sequential dependence of repeated one-step prediction. These approaches reduce prediction cost through token sparsity or a different prediction interface. However, their published designs tie these gains to specialized world-model training or predictor redesign. This limits direct deployment when an existing checkpoint must be reused without retraining or replacing its predictor. Cross-candidate inference instead reuses information between candidate evaluations while keeping the pretrained predictor and its per-step inputs unchanged. **AdaReP** (Cheng et al., 2026) accelerates control by adapting when the planner is called. It monitors mismatch from a cached rollout and an estimate of local dynamics sensitivity to decide when a plan should be recomputed. AdaReP is itself training-free and leaves the underlying world model and planner unchanged. The distinction is therefore the level of computation being controlled: AdaReP reduces planning calls across environment steps, whereas our method allocates full-horizon rollouts within each call under a fixed replanning schedule. Its sensitivity concerns the propagation

Table 2: Score sensitivity and ranking reversals on Wall and PushT. Sensitivity spread is the ratio of the 90th to 10th percentile of R under a single-score perturbation of $+0.1$. Ranking reversals give the fraction of pools in which centered score RMSE and R rank the allocation rules differently at $B{=}60$. Entries are task-averaged estimates with 95% task-bootstrap intervals.

Model / task	Pools	Sensitivity spread	Ranking reversals
DINO-WM / Wall	900	$1.81\times$ [1.77, 1.85]	20.3% [17.9, 22.8]
DINO-WM / PushT	900	$2.22\times$ [2.13, 2.30]	23.0% [19.9, 26.2]

of model mismatch through the dynamics. Ours concerns the effect of candidate score errors on the soft planning update.

Our contribution is training-free cross-candidate inference targeted at the planning update. Given the same candidate pool and soft update, we use selected full-horizon evaluations to reconstruct unobserved scores and allocate further rollouts according to their expected reduction in update uncertainty. Training-free here means that deployment requires no additional world-model training, policy training, or learned selection network. Trend estimation and Gaussian-process conditioning use evaluations from the current planning problem. Full-soft provides the direct full-evaluation reference for this approximation. The controlled comparisons hold the world model and replanning schedule fixed to isolate candidate-evaluation work. The shared objective of faster control thus leads to distinct interventions in model computation, replanning frequency, and allocation within a planning call.

B Additional Diagnostics

Score sensitivity and ranking reversals. We use 900 candidate pools each from Wall and PushT to test how score errors propagate into planning updates, extending Section 2 across Full-soft and our method's trajectories. Under a fixed single-score perturbation, the ratio between the 90th and 10th percentiles of update error is 1.81 on Wall and 2.22 on PushT (Table 2). **The same score perturbation therefore has different consequences across candidates**, supporting the use of update sensitivity in rollout allocation. We also compare update-aware and total-score-variance allocation at $B{=}60$, with posterior-mean reconstruction shared by both methods. Centered score RMSE and update error R reverse their rankings in 20.3% of Wall pools and 23.0% of PushT pools after task averaging. Figure 2 displays these reversals as opposite signs in the log ratios of the two errors. These comparisons motivate selecting evaluations by their effect on the action-distribution update.

Analysis protocol. For Figure 1, we use the first Wall candidate pool in lexicographic order. The three single-candidate examples are nearest to the 10th, 50th, and 90th percentiles of the standardized derivative norm. We evaluate perturbations $\delta \in \{\pm 0.01, \pm 0.1, \pm 0.5, \pm 1\}$ and plot the positive perturbations. For each pool, we sample 64 disjoint candidate pairs with a fixed pool-specific seed and reuse them at amplitudes $\delta \in \{0.1, 0.5, 1\}$. Panel (c) uses $\delta{=}0.1$, with errors $(\delta/\sqrt{2}, \delta/\sqrt{2})$ or $(\delta/\sqrt{2}, -\delta/\sqrt{2})$ assigned to each pair. We select the displayed pairs nearest to the 10th, 50th, and 90th percentiles of derivative-vector cosine similarity. The heatmap samples twelve approximately equally spaced sensitivity ranks and reports the pairwise derivative cosines.

All update errors are computed from unprojected weighted moments, using a fixed prefix-weighted provisional variance within each pool. The allocation comparison uses posterior-mean reconstruction with a shared kernel, initialization, and full-horizon rollout budget. We construct intervals from 2,000 bootstrap resamples of the 50 task clusters, keeping pools from the same task together.

We use the first-order model to guide allocation and evaluate its consequences with exact weighted moments. Under random allocation, its mean relative residual is approximately 16.1% on Wall and 24.5% on PushT. An exploratory calibration check gives empirical coverage below 1% for nominal 95% posterior intervals. Accordingly, we use posterior covariance to prioritize evaluations and set the reconstruction penalty, and obtain experimental uncertainty intervals by resampling tasks.

C Experimental Protocols and Additional Results

C.1 Evaluation and Statistical Units

We evaluate control on 50 tasks with planner seeds $0, 1, 2$ and use the task as the statistical unit. We average seeds and candidate-pool statistics within each task before aggregation, and pair methods by task when estimating differences. Our bootstrap intervals quantify variation across tasks for the fixed checkpoints and planner seeds. We report each comparison separately without a multiple-comparison adjustment.

Task construction and execution. DINO-WM start–goal pairs follow the upstream seed-42 validation split and target-generation procedure, with a separately generated, fixed test manifest. We use ten optimization rounds per call and at most 20 replanning calls for DINO-WM. The horizon counts predictor steps, each of which may represent a block of environment commands under the model's action preprocessing. LeWM uses the released PushT and OGBench-Cube environments, with dataset starts and goals 25 steps later. It uses 30 optimization rounds and a 50-step execution budget. Each plan contains five predicted steps with five commands per step, giving 25 environment commands between replanning calls. We run LeWM quality evaluations in environment batches of two and measure latency at batch size one. Terminal distance is computed between terminal and target states. DINO-WM navigation uses native state coordinates, DINO-WM PushT includes position, angle, and velocity, and LeWM uses the state metric of each released environment. We compare distances only within the same model–task setting.

Controls and reconstruction. Table 3 uses the five soft-update methods from the main comparison. Full-soft evaluates all candidates to the full horizon. Reduced-N uses a smaller pool matched to our prediction work. Uniform and total-score-variance allocation share our GP and uncertainty-penalized reconstruction. Total-score-variance allocation maximizes the reduction in $\mathrm{tr}(C)$, accounting for covariance across candidates. Maximum-variance allocation selects the largest individual C_{ii}. Prefix top-B selects the best prefix evaluations after the common reference and GP initialization.

Comparison structure. We use the main-comparison runs in Table 1 to report success, terminal distance, and latency. The reconstruction experiments in Tables 12 and 13 compare allocation rules on shared candidate pools. Each ablation suite has its own complete-method control, so every component comparison uses runs from the same suite. We count a run once when reusing it across tables or sweeps.

Quality under soft updates. We report success intervals and terminal distances in Table 3 to complement the latency comparison in Table 1. **At matched prediction work, our mean success exceeds Reduced-N's by 3.2–8.6 pp across the five settings.** It also exceeds both Uniform and total-score-variance allocation in every setting. The differences from Full-soft range from -0.7 to $+1.3$ pp. On both LeWM tasks, our method also has lower terminal distance than Reduced-N, Uniform, and total-score-variance allocation, with means of 136.23 on PushT and 0.108 on Cube. Our method has the lowest terminal distance among the five methods on DINO-WM/PushT. On LeWM/PushT and Cube, our terminal distances are within 0.46 and 0.002 of Full-soft's, respectively.

C.2 Control Metrics and Execution Length

Definitions and units. For each episode, we compute action-variation RMS from first differences and second-difference RMS from second differences over time and action coordinates. We average these episode statistics across the three seeds within each task and bootstrap the 50 task means. We compute both differences in command space, using native command units and the task's execution frequency, and compare methods within each model–task setting. Saturation is the fraction of command coordinates at or beyond the outer 1% of a finite action range. For DINO-WM, we evaluate saturation on PointMaze using its finite command bounds. We mark the Wall and PushT entries with dashes because their command representations do not provide the finite bounds required by this metric.

Table 3: Success rates (SR, %) and terminal distances for the five soft-update methods in Table 1. Brackets give marginal 95% task-bootstrap intervals over 50 tasks, with the three seeds kept together. We measure terminal distance in each task's native state units and compare methods within each model–task setting.

Model / task	Method	SR (%) ↑	Terminal distance ↓
DINO / PointMaze	**Ours**	61.3 [49.3, 73.3]	4.57 [4.41, 5.55]
	Full-soft	61.7 [49.0, 75.0]	4.86 [4.35, 5.40]
	Reduced-N	52.7 [40.7, 65.3]	4.90 [4.37, 5.46]
	Uniform	54.0 [47.3, 58.7]	4.87 [3.90, 4.86]
	Total-score variance	52.3 [41.0, 63.0]	4.83 [4.01, 5.08]
DINO / Wall	**Ours**	58.0 [44.7, 71.3]	8.99 [6.89, 11.24]
	Full-soft	58.7 [45.3, 72.0]	8.57 [6.48, 10.78]
	Reduced-N	54.7 [42.0, 67.3]	9.25 [7.15, 11.45]
	Uniform	55.3 [42.0, 68.7]	9.08 [6.97, 11.32]
	Total-score variance	56.0 [42.7, 69.3]	9.05 [6.97, 11.28]
DINO / PushT	**Ours**	27.3 [16.0, 39.3]	82.33 [71.81, 93.68]
	Full-soft	26.0 [15.3, 38.0]	85.02 [74.29, 96.55]
	Reduced-N	24.1 [13.4, 36.1]	107.91 [97.58, 118.87]
	Uniform	22.0 [10.7, 34.0]	83.17 [72.65, 94.67]
	Total-score variance	25.3 [14.0, 37.3]	84.00 [73.82, 95.12]
LeWM / PushT	**Ours**	36.7 [27.3, 46.7]	136.23 [130.90, 163.22]
	Full-soft	36.0 [26.0, 46.7]	135.77 [118.93, 154.08]
	Reduced-N	32.0 [22.0, 42.0]	138.37 [119.80, 158.44]
	Uniform	25.0 [15.7, 35.0]	149.81 [134.93, 166.24]
	Total-score variance	31.3 [20.7, 42.0]	138.18 [122.87, 153.93]
LeWM / Cube	**Ours**	53.3 [40.0, 67.3]	0.108 [0.082, 0.138]
	Full-soft	53.3 [40.0, 66.7]	0.106 [0.077, 0.136]
	Reduced-N	48.0 [34.7, 60.7]	0.109 [0.078, 0.143]
	Uniform	49.2 [35.9, 62.6]	0.110 [0.081, 0.142]
	Total-score variance	50.3 [37.0, 63.0]	0.110 [0.082, 0.141]

Execution length on common successes. Each recorded DINO-WM planner step executes five primitive environment commands. Table 5 compares our method with prediction-work-matched Reduced-N on task–seed pairs where both methods succeed. For each eligible pair, we subtract Reduced-N's execution length from ours and average the differences in primitive steps. We form 95% intervals from 10,000 bootstrap resamples of the 50 task clusters, retaining their planner seeds and excluding resamples without eligible pairs. This pairing compares execution lengths over the same successful episodes. We report the paired intervals alongside mean differences of -0.36 to $+1.39$ primitive steps in Table 5.

LeWM control measurements. We collect LeWM command statistics in separate runs on the same 50 tasks and planner seeds, using the checkpoint and planner configuration of the main comparison. Table 6 pairs the success rate of each run with its command statistics over the evaluation window. Our action-variation RMS and second-difference RMS are lower than those of Uniform and total-score-variance allocation on both tasks. On Cube, our method also has the highest mean success in this suite, at 52.7%, with saturation of 2.97%.

Table 4: DINO-WM control metrics for the five soft-update methods. Action-variation RMS and second-difference RMS are measured in native command units. Downward arrows indicate less variation, oscillation, or saturation. Brackets give 95% task-bootstrap intervals. We report saturation in percent using finite command bounds on PointMaze. Dashes mark omitted entries.

Method	Action-variation RMS ↓	Second-difference RMS ↓	Saturation (%) ↓
DINO-WM / PointMaze			
Ours	0.173 [0.142, 0.207]	0.340	0.10
Full-soft	0.162 [0.156, 0.168]	0.279	0.00
Reduced-N	0.211 [0.203, 0.219]	-	-
Uniform	0.505 [0.475, 0.536]	0.866	0.82
Total-score variance	0.308 [0.284, 0.334]	0.526	0.50
DINO-WM / Wall			
Ours	0.416 [0.411, 0.421]	0.717	-
Full-soft	0.413 [0.409, 0.418]	0.713	-
Reduced-N	0.531 [0.521, 0.540]	0.916	-
Uniform	0.417 [0.411, 0.421]	0.719	-
Total-score variance	0.413 [0.408, 0.418]	0.713	-
DINO-WM / PushT			
Ours	0.119 [0.116, 0.123]	0.206	-
Full-soft	0.115 [0.112, 0.117]	0.197	-
Reduced-N	0.145 [0.141, 0.148]	0.250	-
Uniform	0.121 [0.118, 0.124]	0.208	-
Total-score variance	0.120 [0.117, 0.122]	0.206	-

Table 5: Paired DINO-WM execution lengths at matched prediction work. We pair our method with Reduced-N on episodes where both succeed and measure length in primitive environment-action steps. Differences are ours minus Reduced-N, so negative values indicate shorter executions. Brackets give 95% task-cluster bootstrap intervals.

Task	Paired episodes	Execution-length difference (steps) ↓	95% interval
PointMaze	55	-0.36	$[-5.33, 3.72]$
Wall	80	$+0.69$	$[-0.69, 2.56]$
PushT	36	$+1.39$	$[-2.39, 5.78]$

Table 6: LeWM control metrics from dedicated command-logging runs of the five soft-update methods. We use the same 50 tasks, three seeds, checkpoint, and planner configuration as the main comparison, and report success for these runs alongside statistics of their executed commands. Brackets give 95% task-bootstrap intervals.

Method	SR (%) ↑	Action-variation RMS ↓	Second-difference RMS ↓	Saturation (%) ↓
LeWM / PushT				
Ours	33.3 [24.0, 43.3]	0.200 [0.194, 0.206]	0.318	0.09
Full-soft	36.0 [26.0, 46.7]	0.162 [0.136, 0.187]	0.280	0.01
Reduced-N	31.3 [21.3, 41.3]	0.163 [0.138, 0.188]	0.282	0.03
Uniform	32.0 [22.7, 42.0]	0.294 [0.287, 0.302]	0.508	0.11
Total-score variance	31.3 [30.7, 34.0]	0.293 [0.286, 0.300]	0.505	0.11
LeWM / Cube				
Ours	52.7 [39.3, 66.0]	0.498 [0.491, 0.505]	0.792	2.97
Full-soft	50.3 [37.0, 63.0]	0.439 [0.384, 0.490]	0.760	2.55
Reduced-N	50.3 [37.0, 63.0]	0.453 [0.405, 0.498]	0.784	3.59
Uniform	51.3 [38.0, 64.7]	0.595 [0.587, 0.602]	1.027	3.54
Total-score variance	50.7 [37.3, 63.3]	0.605 [0.596, 0.615]	1.048	3.68

Table 7: Planning latency in seconds: mean, median (P50), and 95th percentile (P95). Percentiles summarize call durations. DINO-WM uses 300 observation-to-action calls per method, and LeWM uses 100 solver calls per method. Timing scopes are defined in Appendix C.3.

	Full-soft: Time (s)			Ours: Time (s)		
Model / task	Mean ↓	P50 ↓	P95 ↓	Mean ↓	P50 ↓	P95 ↓
DINO / PointMaze	9.752	9.745	9.851	6.092	6.080	6.279
DINO / Wall	5.069	5.062	5.111	3.266	3.261	3.292
DINO / PushT	14.418	14.406	14.466	6.060	6.051	6.065
LeWM / PushT	3.493	3.484	3.495	2.036	2.004	2.107
LeWM / Cube	3.490	3.481	3.490	2.077	2.071	2.143

Table 8: Additional Wall comparisons on 50 tasks and three seeds. Brackets give 95% task-bootstrap intervals. Prefix top-B and maximum-variance allocation share our reconstruction model and budget of 780 candidate prediction steps per round. The last two rows reduce our optimization-round count or horizon. Latencies use separate H200 timing batches of 300 observation-to-action calls per method, with the same scope as the main comparison.

Method / configuration	SR (%) ↑	Terminal distance ↓	Time (s) ↓
Ours	58.0 [44.7, 71.3]	8.99	3.266
Prefix top-B	57.3 [44.7, 69.3]	8.65	3.318
Max variance	56.0 [42.7, 68.7]	9.22	3.341
Ours: five rounds	42.0 [28.7, 55.3]	11.54	1.673
Ours: $H = 3$	35.3 [22.7, 48.7]	12.58	2.916

C.3 Timing Details

DINO-WM timing uses ten warm-up calls per method, followed by three repetitions on each of 100 shared start–goal pairs. We synchronize CUDA before and after each observation-to-action measurement and interleave methods, tasks, and repetitions using a fixed random seed. Timing includes all planning and inference operations. It excludes workspace creation, checkpoint loading, environment stepping and resetting, video, and diagnostic logging. All main DINO-WM timing comparisons use NVIDIA H200 GPUs. For speedup intervals, we average the three repetitions within each task, jointly resample the 100 task indices for both methods, and take the ratio of mean latencies. These intervals quantify timing variability under the measured hardware and software configuration.

For LeWM, we time two solver calls per task along each planner's evaluation trajectory at seed 0, giving 100 calls across 50 tasks. Each measurement runs from transformed observations to an action plan. Table 7 reports mean, P50, and P95 latency within each model–task setting.

C.4 Additional Wall Baselines

Update-aware allocation has the highest mean success among the three allocation rules at the same prediction budget. With the reconstruction model fixed, our method reaches 58.0% success on Wall, compared with 57.3% for prefix top-B and 56.0% for maximum-variance allocation (Table 8). The corresponding gains are 0.7 and 2.0 pp. We also vary optimization depth and horizon to distinguish selective evaluation from shortening the search. Five optimization rounds give 42.0% success at 1.673 seconds, and a horizon of three gives 35.3% success at 2.916 seconds. Our default retains the longer search and exceeds these success rates by 16.0 and 22.7 pp, respectively.

C.5 Budget, Prefix, and Search-Size Sweeps

We vary continuation budget, prefix length, and candidate-pool size to examine how prediction work should be distributed within a planning call. Tables 9 and 10 report the budget and horizon sweeps analyzed in Appendix D.6.

Table 9: Success rate (%) of our method across continuation budgets, with 95% task-bootstrap intervals in brackets. All settings use N=300, H=5, h=2, and 150 episodes. The B=60 column matches Table 1. For PushT, we show budgets from B=30 to B=180.

	SR (%) ↑ at continuation budget B				
Model / task	15	30	60	120	180
DINO-WM / Wall	53.0 [31.2, 68.5]	57.0 [45.9, 70.7]	58.0 [44.7, 71.3]	58.7 [45.3, 72.0]	58.0 [44.7, 71.3]
DINO-WM / PushT	-	25.8 [18.7, 40.7]	27.3 [16.0, 39.3]	28.3 [14.7, 36.7]	28.7 [16.0, 38.0]
LeWM / Cube	50.7 [41.3, 67.3]	51.3 [38.0, 64.0]	53.3 [40.0, 67.3]	54.7 [41.3, 68.0]	53.3 [40.7, 66.0]

Table 10: Prefix and horizon settings for our method on DINO-WM / Wall (Figure 13). Candidate prediction steps are counted per optimization round, with work matched separately within each horizon. Brackets give 95% task-bootstrap intervals. Configurations shared by the fixed-budget and fixed-work comparisons use the same run. All settings use N=300, ten optimization rounds, and 150 episodes.

Comparison	H	h	B	Candidate prediction steps / round ↓	SR (%) ↑
Fixed B	5	2	60	780	58.0 [42.7, 69.3]
Fixed B	5	3	60	1020	58.0 [42.7, 69.3]
Fixed B	10	2	60	1080	71.3 [60.0, 82.0]
Fixed B	10	4	60	1560	75.3 [54.0, 86.0]
Fixed B	10	6	60	2040	73.3 [52.0, 84.7]
Matched work	5	2	140	1020	59.7 [45.3, 72.0]
Matched work	5	3	60	1020	58.0 [42.7, 69.3]
Matched work	10	2	180	2040	67.3 [56.0, 78.0]
Matched work	10	4	140	2040	68.7 [58.0, 79.3]
Matched work	10	6	60	2040	73.3 [52.0, 84.7]

On Wall, our mean success increases with candidate-pool size at a fixed continuation fraction of 20%. Increasing N from 100 to 1,200 raises success from 55.3% to 59.3%, a gain of 4.0 pp, while B increases from 20 to 240 (Table 11, Figure 7(a)). The largest increase occurs between N=100 and N=300, where success reaches 58.0%, followed by 58.7% at N=600 and 59.3% at N=1,200. Uniform rises from 50.7% to 56.0% at N=300 and remains at 56.0% for the larger pools. Our mean success exceeds Uniform by 2.0–4.6 pp across the four pool sizes and stays within 0.7 pp of Full-soft at the three sizes where both are reported. At N=600, both our method and Full-soft achieve 58.7% mean success. With H=5, h=2, and B/N=0.2, our method uses $2.6N$ candidate prediction steps per round, compared with $5N$ for full evaluation, retaining a 48% reduction throughout the sweep.

On LeWM/Cube, our success rates at N=100, 300, 600, and 1,200 are 52.67%, 53.33%, 52.00%, and 52.67%, with B/N=0.2 and fixed prefix and horizon lengths. The default N=300 gives our highest mean in this sweep, while Full-soft remains between 52.00% and 53.33% over the same range. This setting combines a 300-candidate update with only 60 full-horizon continuations.

At the default pool size, using 5, 10, 20, and 30 optimization rounds gives success rates of 52.00%, 52.67%, 54.67%, and 52.67%, respectively. We use a separate 30-round control for this sweep. Together, these measurements characterize the success obtained from larger candidate pools and longer optimization schedules, with the frozen model, goal cost, and update rule held fixed.

C.6 Same-Pool Reconstruction Protocol

We collect candidate pools from Full-soft and our method's trajectories at three planner seeds and three optimization stages. The stage indices are 0, 4, 9 for DINO-WM and 0, 14, 29 for LeWM. We select one replanning position per episode using a fixed task–seed rule. This gives $50 \cdot 3 \cdot 2 \cdot 3 = 900$ pools per model–task setting. Each pool contains 300 candidate action sequences with prefix

Table 11: Wall candidate-pool sweep at B/N=0.2, H=5, h=2, and ten optimization rounds. Entries give success rates (%) and 95% task-bootstrap intervals from 150 episodes. Full-soft evaluates all N candidates, while our method and Uniform continue B candidates after the prefix. We use separate control runs for this sweep. Dashes denote settings not evaluated.

		SR (%) ↑		
N	B	Ours	Full-soft	Uniform
100	20	55.3 [42.0, 68.0]	56.0 [42.7, 69.3]	50.7 [38.0, 63.3]
300	60	58.0 [42.7, 69.3]	58.7 [45.3, 72.0]	56.0 [42.7, 69.3]
600	120	58.7 [43.3, 70.0]	58.7 [45.3, 71.3]	56.0 [42.7, 69.3]
1200	240	59.3 [44.0, 70.7]	—	56.0 [42.7, 69.3]

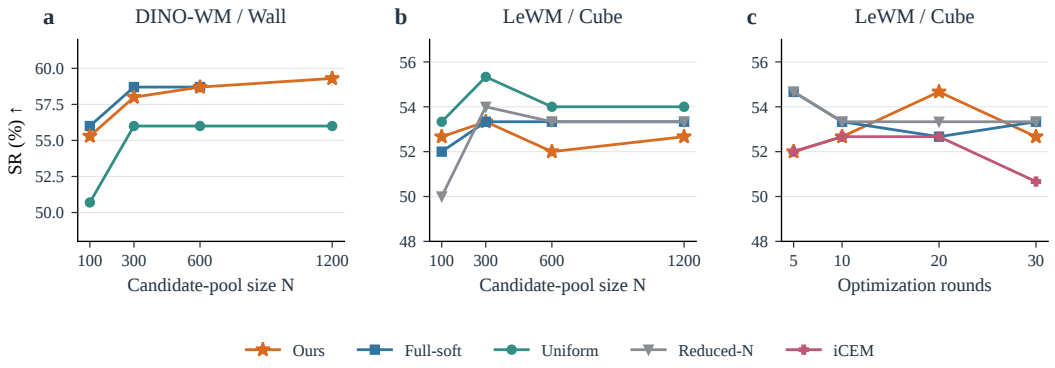


Figure 7: Success across candidate-pool sizes and optimization-round counts. (a) DINO-WM/Wall pool size, with values and intervals in Table 11. (b) LeWM/Cube pool size. (c) LeWM/Cube optimization rounds. Pool-size sweeps use B/N=0.2 and match Reduced-N by prediction work where available. Each point averages 50 tasks and three seeds. Lines connect mean success rates for the evaluated configurations.

and full-horizon evaluations. Every allocation method receives the same pool and 60 full-horizon observations. We average Uniform's three selection seeds within each pool, then aggregate the pool statistics by task for all methods.

Correction RMSE compares $m_i - \frac{1}{2}C_{ii}$ with the observed correction y_i over unobserved candidates $i \notin \mathcal{S}$. The reference subtraction in Eq. (7) removes a common score offset. We compute R from the exact unprojected weighted moments of the recorded actions. Normalization uses the pool's positive prefix-weighted provisional variance $\widetilde{\mathbf{v}}_t$ in Eq. (5), held fixed across allocation methods and used without a variance floor. First-action error is the Euclidean distance between the first weighted-mean action vectors, measured before clipping or execution. The results for Wall, PushT, and Cube are reported together in Appendix D.4.

This protocol directly tests how the posterior covariance guides observation selection and how the reconstructed scores affect the planning update. Appendix B gives the complementary sensitivity and posterior-coverage diagnostics.

D Extended Experimental Analysis

D.1 Extended Setup and Baseline Definitions

World models and tasks. We evaluate released DINO-WM checkpoints on PointMaze, Wall, and PushT (Zhou et al., 2025), and released LeWM checkpoints on PushT and Cube (Maes et al., 2026a). DINO-WM uses spatial patch representations, while LeWM uses a compact global latent representation. All methods within a model–task setting share the frozen encoder and predictor, observation processing, goal cost, and execution budget. Each method is evaluated on 50 fixed start–goal tasks with three planner seeds, giving 150 episodes. DINO-WM tasks follow the upstream generation procedure with a test manifest separate from development tasks. LeWM uses dataset starts and goals 25 environment steps later. Success is averaged over all tasks and seeds using each environment's evaluation criterion. The two PushT evaluations follow their respective world-model protocols.

Planning and baselines. The default setting is $N{=}300$, $B{=}60$, $H{=}5$, and $h{=}2$, with the earlier trend evaluation at $h_1{=}1$. We fix $T{=}1$, kernel length scale $\ell_\phi{=}1$, and kernel amplitude $\sigma_f{=}0.1$ across the main comparisons. DINO-WM uses ten optimization rounds and at most 20 replanning calls, each executing five environment commands. LeWM uses 30 rounds and a 50-step execution budget, with five predicted blocks of five commands executed before replanning. We use Full-soft as the direct reference for cross-candidate inference: it evaluates every candidate to horizon H and applies the same update in Eq. (2). Reduced-N uses the same update with 156 candidates, matching our 780 candidate prediction steps per round. Uniform and total-score-variance allocation share our candidate pool, GP, and reconstruction rule, changing only the selection of full-horizon rollouts. Total-score-variance allocation greedily reduces $\mathrm{tr}(C)$, while our rule reduces $\mathrm{tr}(G^\top C G)$. We include Full-hard and iCEM (Pinneri et al., 2021) in the DINO-WM success–latency plot to place this comparison alongside alternative update and sampling rules. Full-hard updates from the best 10% of candidates after full evaluation, with the same horizon and optimization-round count as Full-soft. iCEM uses temporally correlated sampling, elite reuse, and a decreasing population.

Metrics and intervals. The primary metrics are closed-loop success rate and measured planning latency. We report success differences in percentage points (pp) and speedup as Full-soft's mean latency divided by the compared method's mean latency. Candidate prediction steps count model-evaluation work, and terminal distance uses each task's native state coordinates. Reconstruction is evaluated by the RMSE of unobserved corrections y_i, update error R in Eq. (5), and first-action error. Executed-command statistics and execution length are defined in Appendix C.2. Intervals use 10,000 bootstrap resamples of the 50 task clusters, retaining all three seeds within each task. Paired comparisons resample the same task indices for both methods.

Timing protocol. Experiments use 11 NVIDIA H200 GPUs, with one GPU per latency measurement. DINO-WM timing covers the synchronized observation-to-action call at environment batch size one. Each method uses the same 100 start–goal pairs, three repetitions per pair, and ten warm-up calls, with measurements interleaved in a fixed randomized order. Timing includes planning and inference overhead and excludes model loading, environment physics, resetting, and recording. LeWM timing covers the solver from transformed observations to an action plan at batch size one, using 100 calls across 50 tasks. Speedups are computed within each model–task setting using the same timing scope for both methods. Appendix C.3 gives the sampling and timing details.

D.2 Planning Quality and Latency with DINO-WM

We measure complete planning calls to quantify the latency benefit of the prediction savings in Section 3.3, including the cost of cross-candidate inference. All methods share the frozen world model and execution budget. Table 1 reports success and latency for the five soft-update methods, and Figure 8 additionally includes Full-hard and iCEM.

Cross-candidate inference gives a measured 1.55–$2.38\times$ speedup over Full-soft across the three DINO-WM tasks. On PointMaze, latency decreases from 9.752 to 6.092 seconds, with success changing from 61.7% to 61.3%. On Wall, latency decreases from 5.069 to 3.266 seconds, with

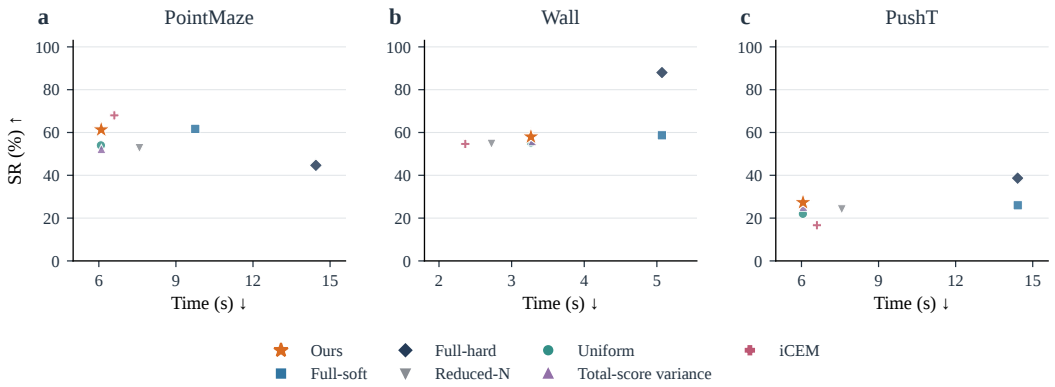


Figure 8: DINO-WM success versus planning latency. The five soft-update methods use the results in Table 1, with success intervals in Table 3. Full-hard and iCEM show the operating points of alternative update and sampling rules. The orange star denotes our method.

success changing from 58.7% to 58.0%. On PushT, latency decreases from 14.418 to 6.060 seconds, with success increasing from 26.0% to 27.3%. The success differences range from -0.7 to $+1.3$ pp.

At the same 780 candidate prediction steps per round, our update retains 300 candidates, compared with 156 for Reduced-N. At this budget, our method improves mean success by 8.6, 3.3, and 3.2 pp on PointMaze, Wall, and PushT, respectively. Our latency is also lower on PointMaze and PushT, decreasing from 7.568 to 6.060 seconds on PushT. On Wall, the two methods occupy different success–latency points: our method achieves 58.0% success at 3.266 seconds, and Reduced-N achieves 54.7% at 2.723 seconds. We report both prediction work and latency because candidate batching and reconstruction contribute differently to complete-call cost.

The Full-soft comparison holds the update rule fixed and measures the benefit of sharing information across candidate evaluations. We then use reconstruction, ablation, and budget experiments to examine how the selected observations affect this update and the resulting control behavior.

D.3 Action Regularity and Execution Length

We evaluate executed commands using first- and second-difference RMS, saturation where finite bounds apply, and execution length. These measurements complement the first-action reconstruction error by following the executed control trajectories. Figure 9 pairs action-variation ratios with the success differences in Table 1. Definitions and complete control statistics appear in Appendix C.2.

Cross-candidate inference reduces mean action variation relative to Reduced-N on all three DINO-WM tasks at the same prediction budget. On Wall, action-variation RMS is 0.4159 for our method, 0.5306 for Reduced-N, and 0.4134 for Full-soft. The reduction relative to Reduced-N is 21.6%. On PushT, the corresponding values are 0.1194, 0.1446, and 0.1146, giving a 17.4% reduction. Paired 95% intervals for our ratios to Reduced-N are $[0.776, 0.792]$ and $[0.814, 0.838]$, respectively. Second-difference RMS is also lower than Reduced-N's on both tasks (Table 4). These reductions accompany the success gains over Reduced-N under the same prediction-work budget.

On PointMaze, action-variation RMS is 0.173 for our method and 0.211 for Reduced-N, a reduction of 18.0%. Our method also has lower first- and second-difference RMS than Uniform and total-score-variance allocation, with saturation of 0.10% versus 0.82% and 0.50%, respectively. The separate LeWM runs also show lower first- and second-difference RMS than Uniform and total-score-variance allocation on both tasks (Table 6). On Cube, our method has the highest mean success rate among the five methods, at 52.7%, and lower saturation than Reduced-N, at 2.97% versus 3.59%. We also compare execution lengths on task–seed pairs where both methods succeed. Our mean differences from Reduced-N are -0.36, $+0.69$, and $+1.39$ primitive steps on PointMaze, Wall, and PushT, respectively, with paired intervals in Table 5. We use this comparison to distinguish the number of executed commands from the computation required to plan each command sequence.

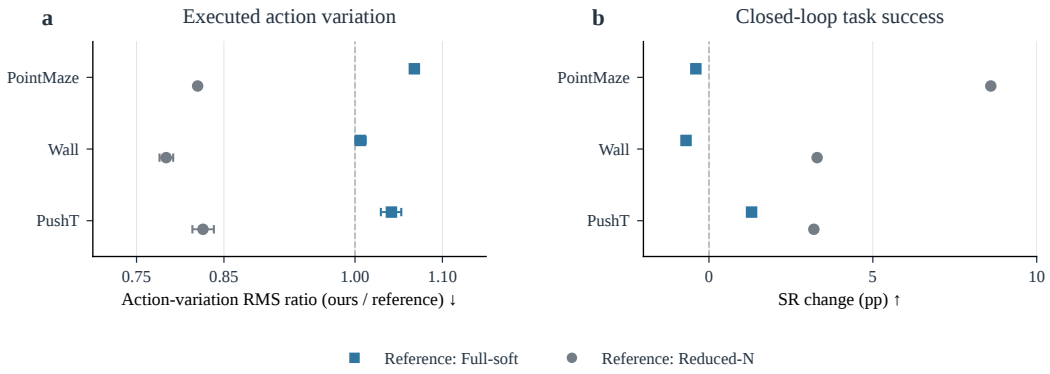


Figure 9: Action variation and success on DINO-WM, relative to Full-soft and prediction-work-matched Reduced-N. (a) Action-variation RMS ratios, with PointMaze values from Table 4 and paired 95% task-bootstrap error bars on Wall and PushT. Ratios below one indicate smoother commands from our method. (b) SR changes from Table 1, computed as ours minus reference in percentage points (pp). Positive values favor our method.

Table 12: Wall and PushT reconstruction at B=60, with Eq. (15) shared by all allocation methods. Correction RMSE measures unobserved, reference-relative correction errors. Update error R uses unprojected moments and the prefix-variance normalization in Eq. (5). First-action error is measured before clipping. Each task setting contains 900 pools across 50 tasks. Uniform's three selections are averaged within each pool.

	DINO-WM / Wall			DINO-WM / PushT		
Allocation	Correction RMSE ↓	Update error R ↓	First-action error ↓	Correction RMSE ↓	Update error R ↓	First-action error ↓
Ours	0.167	0.0148	0.0314	0.307	0.0286	0.0554
Uniform	0.1941	0.0180	0.0405	0.3416	0.0329	0.0719
Total-score variance	0.191	0.0196	0.0396	0.351	0.0306	0.0574
Max variance	0.188	0.0204	0.0373	0.334	0.0311	0.0590
Prefix top-B	0.3710	0.0351	0.1084	0.4595	0.0435	0.1205

D.4 From Reconstructed Scores to Planning Updates

We compare reconstruction on 900 fixed candidate pools each from DINO-WM/Wall, DINO-WM/PushT, and LeWM/Cube. The pools cover 50 tasks, three planner seeds, two trajectory sources (Full-soft and our method), and three optimization stages. We hold the candidate actions, prefix evaluations, and full-horizon reference scores fixed across allocation methods. At B=60, we compare update-aware allocation (Ours, Eq. (13)) with Uniform, total-score-variance, maximum-variance (Max variance), and prefix top-B allocation. All methods share the final uncertainty-penalized reconstruction rule. Uniform uses three selection seeds, averaged within each pool before task aggregation. Appendix C.6 defines the pool collection and error metrics.

Our allocation rule gives the lowest correction RMSE, update error R, and first-action error among the compared methods on both DINO-WM tasks. Relative to Uniform, our method reduces R by 17.8% on Wall and 13.1% on PushT, with first-action error lower by 22.5% and 22.9%, respectively (Table 12, Figure 10). Correction RMSE decreases by 14.0% on Wall and 10.1% on PushT. Because we share the candidate pools, model, budget, and reconstruction rule, these comparisons isolate the effect of rollout allocation.

On Wall, maximum-variance and total-score-variance allocation both lower correction RMSE relative to Uniform, yet increase R by 13.3% and 8.9%, respectively. On PushT, maximum-variance allocation has lower correction RMSE than total-score-variance allocation (0.334 versus 0.351), but higher R (0.0311 versus 0.0306). This ranking reversal agrees with Section 2: the effect of score errors depends on their joint direction and the update sensitivities.

Table 13: LeWM/Cube reconstruction on 900 candidate pools, using the protocol and metrics of Table 12. Lower values indicate smaller errors. We interpret R within Cube because its normalization uses the model's prefix-weighted action variances.

Allocation	Correction RMSE ↓	Update error R ↓	First-action error ↓
Ours	7.758	1.45×10^{29}	1.010
Uniform	9.304	4.62×10^{29}	1.918
Total-score variance	8.299	5.69×10^{29}	1.868
Max variance	8.728	7.16×10^{22}	1.434
Prefix top-B	15.340	1.48×10^{27}	1.899

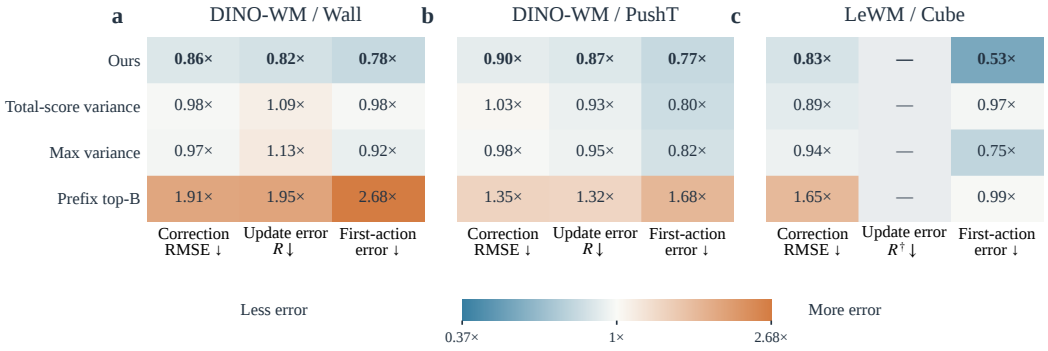


Figure 10: Reconstruction errors relative to Uniform on fixed candidate pools. Cells report ratios of task-averaged errors, with Uniform equal to one and lower values indicating less error. Cell colors use a logarithmic scale, while printed values show the ratios. For Cube, we compare correction RMSE and first-action error in the heatmap and report update error R separately in Table 13 (†), since prefix variances near 10^{-34} dominate its normalization. Ratios use the values in Tables 12 and 13.

Prefix top-B gives larger correction and update errors on both tasks. Its first-action errors are 0.1084 on Wall and 0.1205 on PushT, compared with 0.0314 and 0.0554 for our method. This pattern supports selecting observations that inform the weighted update across the pool, including candidates outside the best prefixes.

Reconstruction on Cube. Our method also achieves the lowest correction RMSE and first-action error among the compared allocation methods on LeWM/Cube (Table 13, Figure 10(c)). Relative to Uniform, correction RMSE decreases from 9.304 to 7.758, a reduction of 16.6%, and first-action error decreases from 1.918 to 1.010, a reduction of 47.3%. Our first-action error is also 29.6% lower than the next-best value of 1.434 from maximum-variance allocation.

For Cube, we use first-action error to compare the proposed commands directly, alongside the normalized update error in Table 13. Prefix-weighted action variances reach 1.11×10^{-34}, amplifying R to values between 7.16×10^{22} and 5.69×10^{29}. We therefore present correction and first-action error ratios in Figure 10(c) and keep the full R values in the table.

We use the fixed-pool experiments to measure how closely cross-candidate inference reproduces a full-evaluation update. The closed-loop comparisons in Table 1 then evaluate the resulting behavior over complete task trajectories.

D.5 Component Ablations

We ablate one component at a time with the world model and rollout budget fixed. State only removes the remaining-control features from ϕ_i. No control spectrum retains the control mean but removes the covariance-spectrum features. Posterior mean only removes $-\frac{1}{2}C_{ii}$ from Eq. (15). No prefix trend removes $\beta\boldsymbol{\tau}$ from the GP. No reconstruction uses full-horizon scores for selected

Table 14: Component ablations at N=300, B=60, H=5, and h=2, with 150 episodes per setting. We compare each variant with its suite's complete-method control. Dashes denote variants not evaluated for that task. Distances use task-specific units. Figure 11 shows success differences and their available paired intervals.

	DINO-WM / Wall		DINO-WM / PushT		LeWM / Cube	
Variant	SR (%) ↑	Terminal distance ↓	SR (%) ↑	Terminal distance ↓	SR (%) ↑	Terminal distance ↓
Complete method	56.67	9.08	25.33	83.89	53.33	0.1115
No reconstruction	33.33	12.96	24.67	85.33	—	—
State only	48.00	10.78	22.67	97.14	51.33	0.1157
No control spectrum	—	—	24.67	84.07	52.00	0.1114
Posterior mean only	55.33	10.79	—	—	52.67	0.1131
No prefix trend	55.67	9.90	25.33	84.61	—	—

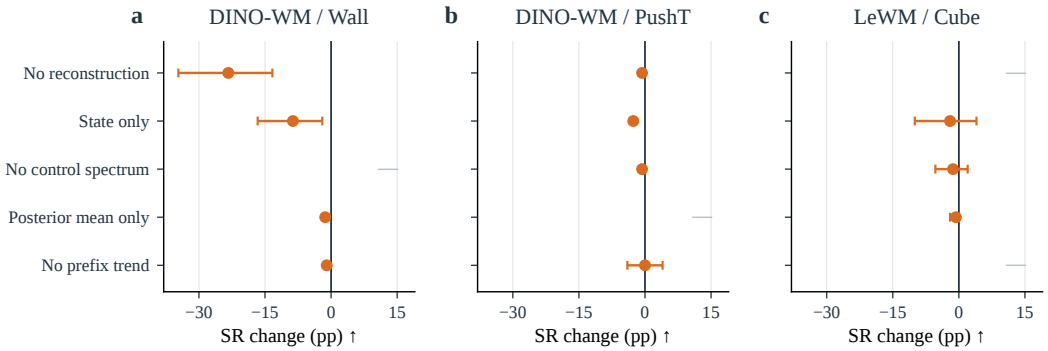


Figure 11: SR changes after ablation at a fixed rollout budget. Each point is the variant minus its complete-method control, in percentage points (pp). Horizontal bars show available paired 95% task-bootstrap intervals. Negative values indicate a loss after ablation. Grey dashes denote variants not evaluated for that task.

candidates and prefix scores for the rest. Table 14 reports the component comparisons, and Figure 11 expresses each success change relative to the complete-method control from the same suite.

Reconstructing unobserved candidates gives the largest ablation effect on Wall. Removing reconstruction lowers Wall success from 56.67% to 33.33%, a loss of 23.33 pp with a paired 95% interval of $[13.33, 34.67]$ pp. Terminal distance increases from 9.08 to 12.96. Both variants retain the same selected full-horizon evaluations, so the difference comes from using those observations to correct the remaining candidates. A short prefix can favor actions whose immediate predictions approach the goal despite poor longer-term outcomes. Reconstruction transfers the observed long-horizon corrections to such candidates before the update. On PushT, success decreases from 25.33% to 24.67%, a smaller loss of approximately 0.7 pp. Terminal distance increases from 83.89 to 85.33, with a paired 95% interval of $[-3.68, 6.72]$ for the difference. Both tasks favor reconstruction at the reported means, with a larger effect on Wall.

Remaining-control features distinguish continuations with similar prefix endpoints. On Wall, state-only features reduce success by 8.67 pp, with a paired 95% interval of $[2.00, 16.67]$ pp, and increase terminal distance from 9.08 to 10.78. This agrees with the construction in Section 3.1, where the remaining controls help determine the long-horizon correction. On PushT, state-only features also lower success (22.67% versus 25.33%) and increase terminal distance (97.14 versus 83.89). On Cube, the complete method reaches 53.33% success, compared with 51.33% for state only and 52.00% without the control spectrum.

The prefix trend improves Wall success from 55.67% to 56.67% and reduces terminal distance from 9.90 to 9.08. PushT success is unchanged at 25.33%. Posterior-mean-only reconstruction gives 55.33% success and a terminal distance of 10.79 on Wall. On Cube, the success difference is -0.67 pp. Removing the control spectrum on PushT lowers success from 25.33% to 24.67% and increases

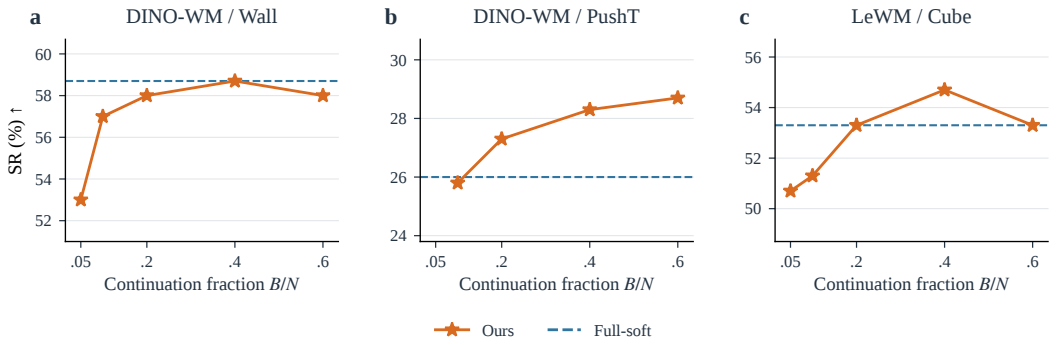


Figure 12: Mean success of our method versus the continuation fraction B/N. Solid curves connect mean success rates, and dashed lines show the corresponding Full-soft references. Values and 95% task-bootstrap intervals appear in Table 9. The PushT curve starts at B=30.

terminal distance from 83.89 to 84.07. The largest success changes occur when we remove reconstruction or the remaining-control features on Wall. The trend, control spectrum, and uncertainty penalty produce smaller changes, consistent with their roles in estimating and weighting the reconstructed corrections.

D.6 Allocating Prediction Work Across Candidates and Time

We vary our continuation budget B over $15, 30, 60, 120$, and 180, with N=300, H=5, and h=2 fixed. These budgets use $43\%, 46\%, 52\%, 64\%$, and 76% of Full-soft's prediction work, respectively. Each task uses its corresponding Full-soft result in Table 1 as the reference.

At the default B=60, success is within 1.4 pp of the highest reported mean on each task, using 52% of Full-soft's prediction work (Table 9, Figure 12). On Wall, success rises from 53.0% at B=15 to 57.0% at B=30 and 58.0% at B=60. Increasing B to 120 gives 58.7%, matching Full-soft's mean success. Thus, most of the observed improvement on Wall is obtained with at most 20% of candidates continued to the full horizon.

On PushT, success increases from 25.8% at B=30 to 27.3% at B=60 and 28.7% at B=180, compared with Full-soft's 26.0%. On Cube, success rises from 50.7% at B=15 to 53.3% at B=60, matching Full-soft's mean, and reaches 54.7% at B=120. Raising B from 60 to 180 increases prediction work from 52% to 76% of full evaluation, with success changes of 0.0, 1.4, and 0.0 pp on Wall, PushT, and Cube, respectively.

We vary prefix length on Wall at H=5 and H=10, either fixing B=60 or adjusting B to hold prediction work constant (Figure 13). **At H=10, the highest reported mean uses h=4 with fixed B and h=6 with fixed prediction work.** With B=60, prefix lengths h=2, 4, 6 give success rates of 71.3%, 75.3%, and 73.3%, using 1,080, 1,560, and 2,040 predictions, respectively. The four-step prefix gives the highest mean success with less work than the six-step prefix. At fixed work of 2,040 predictions, (h, B)=$(2, 180), (4, 140)$, and $(6, 60)$ give 67.3%, 68.7%, and 73.3% success, respectively. Here a longer prefix reduces the number of candidates that can be fully evaluated, yet the six-step prefix achieves the highest mean, 6.0 pp above the two-step prefix. The gain is consistent with more informative prefix features and scores improving selection and reconstruction under the same prediction budget.

At H=5 with B=60, a two-step prefix achieves 58.0% mean success using 780 predictions, matching the three-step prefix's success at 1,020 predictions. With work fixed at 1,020 predictions, (h, B)=$(2, 140)$ gives 59.7% success, compared with 58.0% for $(3, 60)$. The reported means favor more continuations at H=5 and a longer prefix at H=10 under their respective fixed-work budgets. Prefix length and continuation count therefore need to be selected jointly for each horizon and work budget.

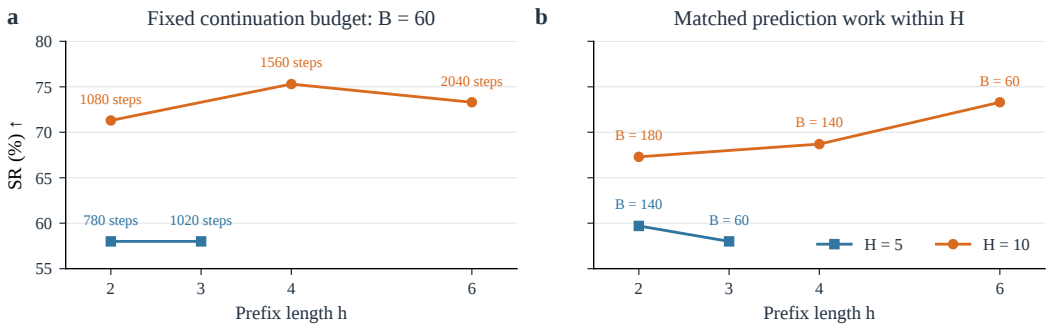


Figure 13: Mean success for prefix and continuation settings of our method on Wall. Left: fixed B=60, with annotations showing prediction steps per round. Right: fixed work of 1,020 predictions at H=5 and 2,040 at H=10, with annotations showing B. All settings use N=300 and ten optimization rounds. Shared configurations use the same run in both panels. Values and 95% intervals appear in Table 10.

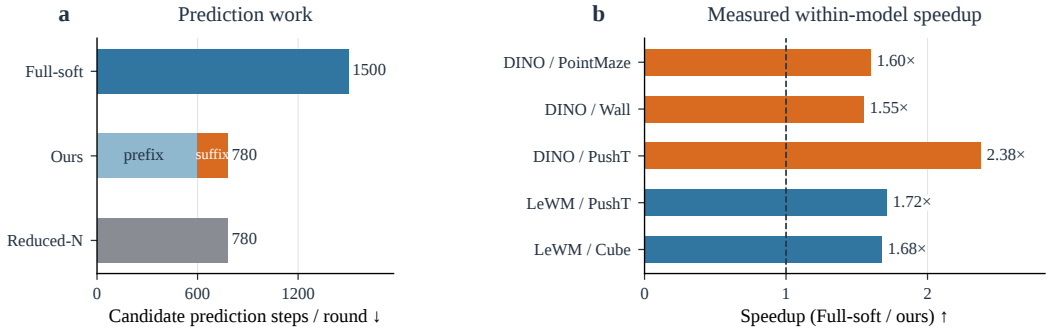


Figure 14: Prediction work and measured speedup. Left: candidate prediction steps per optimization round at the default setting, accounting for prefix reuse. Right: Full-soft's mean latency divided by our mean latency. The dashed line marks equal latency. DINO-WM uses observation-to-action timing, and LeWM uses solver timing. Ratios are computed within each model–task setting.

D.7 Prediction Work and Planning Latency

At the default setting, each update uses $300{\cdot}2{+}60{\cdot}(5{-}2){=}780$ candidate prediction steps, compared with $300{\cdot}5{=}1{,}500$ for Full-soft. This saves 48% of prediction work, while measured DINO-WM latency decreases by 35.57%–57.97% (Figure 14). We measure the complete planning call, including encoding, sampling, feature construction, selection, GP conditioning, and distribution updates. This timing accounts for the inference overhead that accompanies selective rollout evaluation, as well as the throughput of the resulting prediction batches.

Median and tail latency. Both P50 and P95 latency are lower than Full-soft's in all five settings, with cross-candidate inference costs included (Table 7). On DINO-WM, the reductions range from 35.6% to 58.0% at P50 and from 35.6% to 58.1% at P95. On LeWM, P95 latency decreases from 3.495 to 2.107 seconds on PushT and from 3.490 to 2.143 seconds on Cube, reductions of 39.7% and 38.6%, respectively.