DO LATENT WORLD MODELS REALLY SELECT THE RIGHT ACTIONS?

Anonymous authors
Paper under double-blind review

ABSTRACT

Latent world models have recently shown strong results in planning from images, particularly on navigation and manipulation tasks. They score candidate action sequences by the distance between their predicted embeddings and the embedding of a goal image. We find that on fixed candidate sets this score usually misses the best action sequence, and that on the manipulation tasks most models do no better than a random choice. On Push-T the released model's predictor causes this. If these scores misrank fixed candidates, why does planning still work so well? Part of the answer is that the same scores rank the planner's own candidates far above a random choice, with the best one on top in 18.1% of Push-T searches against 1.6%, and we show that wider search and more frequent replanning also contribute to this success. PointMaze success falls from 91% to 76% without replanning. On Push-T, a wider search raises success without replanning from 13% to 67%. Further analysis on PointMaze finds a successful action sequence among the planner's candidates in over 90% of episodes, yet the candidate the model scores highest succeeds in 63%. Our findings suggest that strong results in planning do not show that a latent world model selects the right actions.

1 INTRODUCTION

World models help agents plan by predicting the consequences of their actions. For visual tasks, latent world models encode images into learned representations and predict how these representations change under candidate action sequences (Zhou et al., 2024). The planner scores each action sequence by how close its predicted outcome is to the goal image in representation space. This allows the same learned model to support different goals, each specified by an image of the desired outcome (Zhou et al., 2024). This approach has achieved strong results on navigation and manipulation tasks (Zhou et al., 2024; Terver et al., 2025).

A useful action score should favor sequences that lead to better outcomes in the environment. Task success reflects both the world model and the planner, leaving open how well the model tells good action sequences from poor ones. We examine this by comparing model scores with the outcomes of executing the same candidate action sequences. Despite these strong planning results, we find that latent-space scores often favor poor action sequences (Figure 1a). On fixed candidate sets from the manipulation tasks, most models rank action sequences no better than a random choice. The pattern holds for image- and video-pretrained encoders, but the released Push-T model ranks above random once its predictor is bypassed. Why, then, does planning with these scores work so well?

The same scores rank the planner's own candidates far above random, putting a best one first in 18.1% of Push-T searches against 1.6% (Appendix E). The planner also searches widely over action sequences and revises plans as new observations arrive. On Push-T, a larger search budget raises success from 13% to 67% with the same world model, even when the agent executes its entire plan without replanning (Figure 1b). Replanning lets the agent use new observations to revise a poor plan. The agent executes only the first part of its plan, observes the resulting state, and plans again. On PointMaze, success rises from 76% when following a single plan to 91% with frequent replanning (Figure 1c).

A failed plan leaves two possibilities: the search found no successful action sequence, or the planner failed to use one it had already found. On PointMaze, the second case accounts for most failures of the model's top-scored candidate. These failures leave room to improve planning by using the

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

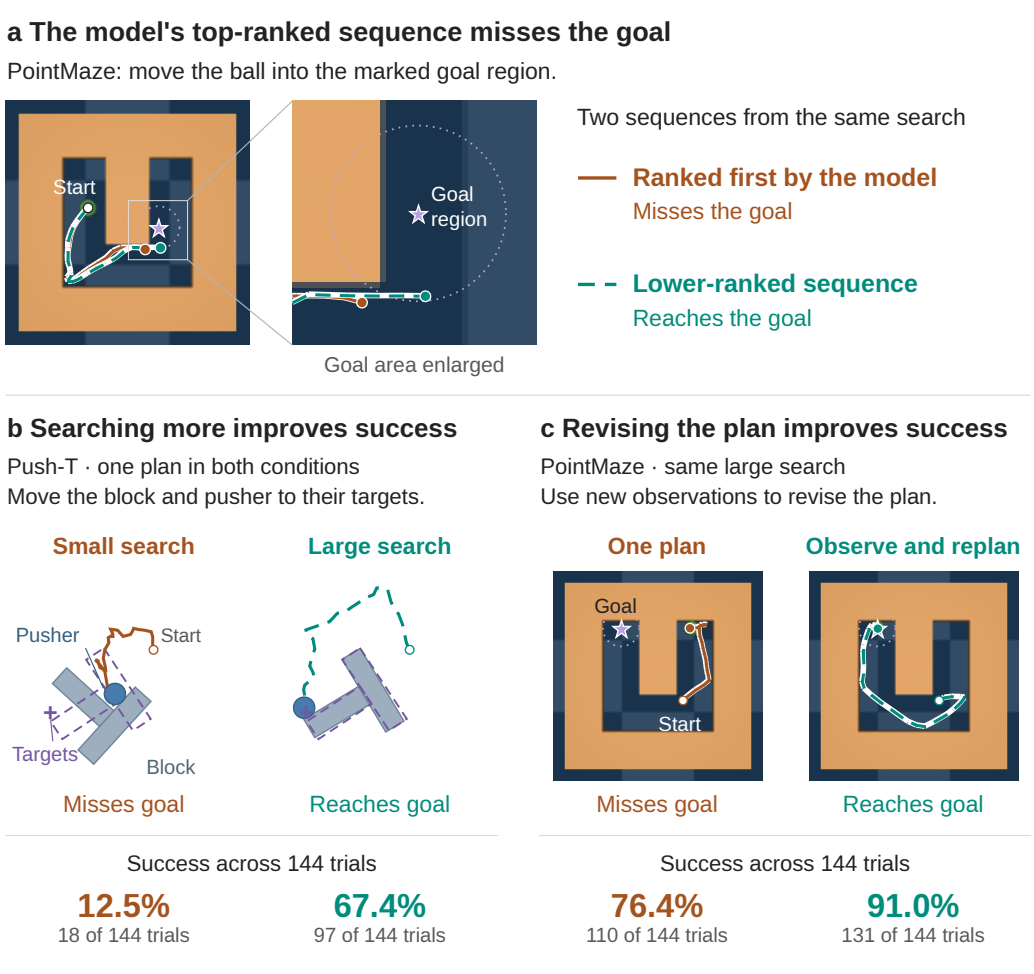


Figure 1: **Planning can succeed even when the model favors unsuccessful action sequences.** (a) In the PointMaze example, the model's top-ranked sequence misses the goal, while a lower-ranked sequence reaches it. (b) A wider search improves Push-T success even when the agent follows a single plan. (c) Replanning from new observations improves PointMaze success at the larger search budget. Purple marks the goal. The paths illustrate individual episodes; success rates summarize 144 paired trials.

action sequences already found. A world model's ability to choose useful actions should therefore be evaluated alongside the success of the planner that uses it.

The main contributions of this work are summarized as follows:

- We show that most models rank fixed candidates no better than random on manipulation tasks, and that on Push-T the released model's predictor causes this.
- We show that wider search and more frequent replanning both contribute to planning success, with their importance varying across tasks.
- We show that planners can fail even when successful action sequences are already present among their candidates.

2 Background and Related Work

Planning with latent world models. An encoder maps an observation to a latent vector, which a predictor then rolls forward under a candidate action sequence. A planner then scores each candidate by the distance between the candidate's predicted latent and the goal observation's latent, before

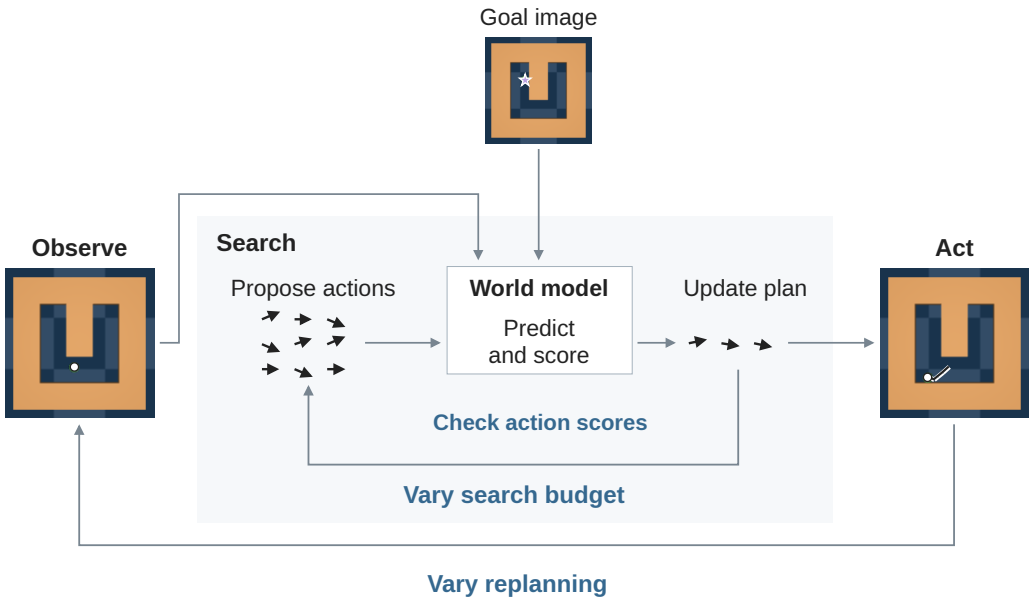


Figure 2: **Search builds a plan; replanning updates it from new observations.** The world model scores candidate action sequences by how close their predicted outcomes are to the goal in representation space. Search repeatedly samples action sequences and uses the average of the best-scored ones to update the plan. The agent then executes the first part of the plan, observes the result, and replans. Blue labels mark where we compare model scores with actual outcomes and test how search and replanning affect success. For failed plans, we also test whether any sequence from the last search round reaches the goal.

executing the first few actions of the resulting plan and replanning from the new observation (Figure 2). DINO-WM, JEPA-WM, V-JEPA 2-AC and related world models use such planners, usually searching over candidates with the cross-entropy method (CEM) (Zhou et al., 2024; Terver et al., 2025; Assran et al., 2025). Each candidate has a true cost, measured from the outcome it reaches rather than the one the model predicts. As a scorer, the latent distance should rank candidates the way their true cost does, and we measure the gap between the two rankings.

Measurements of scorer quality. Scorers are usually judged in one of two ways. First, one line of work scores predicted futures with learned rewards or values, from PlaNet to TD-MPC2 (Hafner et al., 2019; 2020; 2023; Hansen et al., 2022; 2024). There, scorer quality means how well the predicted reward or value tracks the return actually obtained. The reward-free line of work instead ranks candidates by the latent distance and has no learned value, and because a latent distance has no return to compare against, the first way of judging a scorer does not carry over. Second, prior work has measured how much a better model or more search improves control. For example, one-step log-likelihood of model predictions correlates only inconsistently with episode return under sampling-based planning on cartpole and half-cheetah (Lambert et al., 2020). Similarly, evaluation-time search adds only a small boost to MuZero in most environments (Hamrick et al., 2021). Li et al. (2026) instead measure the ranking itself and find that, under a fixed encoder, the latent distance to the goal at a predicted rollout's end misranks candidates, whereas a learned reachability cost ranks the same candidates correctly. The study reports the finding for two reward-free world models, LeWorldModel (LeWM) and PLDM, on TwoRoom and TopoNav. The study also compares scorers on one shared candidate set, with the checkpoint, candidate sampler and search budget fixed, and finds on Push-T that the reachability cost improves candidate ranking more clearly than closed-loop success. We also ask whether the ranking is right, on fixed candidates and in a planner that does not replan, where a wrong choice cannot be corrected at the next step. Because Li et al. (2026) hold the encoder fixed throughout, what a different encoder would do is left open.

Visual encoders. Part of the reward-free line of work takes its encoder from video rather than image pretraining (Bardes et al., 2024; Assran et al., 2025). For policy learning, which encoder works best depends on the learning algorithm (Hu et al., 2023), and training from scratch matched frozen pretrained features (Hansen et al., 2023). For latent planning, swapping the encoder inside DINO-WM changed closed-loop success, and patch features beat a single global feature vector (Zhou et al., 2024). Beyond the encoder, end-to-end variants such as LeWM (Maes et al., 2026) and Delta-JEPA (Zhang et al., 2026) change how the latent space is trained. However, in every study above that swaps or retrains the encoder for control, the evidence is closed-loop success or policy performance, not the ranking itself.

Replanning. Adaptive replanning changes only how often the planner replans, both for latent planners (Cheng et al., 2026) and for map-based navigation (Honda et al., 2024). Three other strategies change the model or the search itself. Temporal straightening retrains the encoder so that latent distance tracks geodesic distance (Wang et al., 2026a), amortised inverse dynamics replaces the search with a learned policy (Nguyen et al., 2026), and test-time adaptation updates the world model while the planner runs (Wang et al., 2026b). The five studies above report closed-loop success or task performance as evidence, and so leave the ranking unmeasured. Closed-loop success cannot stand in for the ranking, because a planner that replans at every step can correct a wrong choice before it costs an episode (Cheng et al., 2026) and so can succeed often even when its ranking is badly wrong. We call this disagreement between the ranking and closed-loop success the ranking–success mismatch. We measure the ranking directly on candidate sets fixed in advance, and compare encoders on one shared candidate set under one scorer. Separately, we run the planner and vary how often it replans, to test whether replanning hides the planner's wrong choices.

3 Experimental Setup

Tasks. JEPA-WM's release covers four simulated tasks and one real-robot dataset with no simulator to compute a true cost, and we take all four. Two of the four tasks are navigation and two are manipulation, and Figure 5 in Appendix A shows an observation and a goal from each. Both kinds of task matter, because a scorer could do well at reaching a position and badly at reaching a pose. PointMaze is from D4RL (Fu et al., 2021), Wall from DINO-WM (Zhou et al., 2024), Push-T from the Diffusion Policy work (Chi et al., 2024), and reach-wall from MetaWorld (Yu et al., 2021). In all four tasks, straight-line distance to the goal misleads a ranking in some situations and not in others, so our measurements record how badly a scorer ranks without separating how much of that comes from the task's geometry.

Models. The primary world models we audit are the four checkpoints that JEPA-WM released for its simulated tasks, one per task, each trained on that task alone. For PointMaze and Wall, the same release also provides DINO-WM checkpoints as reproduced baselines, which we audit under the protocol Appendix E gives. Since JEPA-WM and DINO-WM share the DINOv2 image encoder and differ in the predictor and its action conditioning, any difference in their rankings must come from the predictor. We keep the JEPA-WM and DINO-WM checkpoints frozen, never training, fine-tuning or otherwise changing a weight, so that we measure the released models' ranking. We score with each model's planning objective, the form of the latent distance of Section 2 that Appendix A lists per task, except for a proprioceptive weight of 1 rather than the released 0.1 on PointMaze and Wall. Where a later section trains a scorer, it is trained on the frozen checkpoint's outputs and leaves the checkpoint unchanged. Unlike the JEPA-WM checkpoints, V-JEPA 2-AC is released only for the real-robot dataset that the previous paragraph set aside, so we do not audit it. To examine the encoder's role in action ranking, we compare the image-pretrained DINOv2 with four video-pretrained encoders from V-JEPA 1, V-JEPA 2 and V-JEPA 2.1. On Push-T and MetaWorld reach-wall, we freeze each encoder and train a separate six-layer action-conditioned predictor for each task. We also evaluate LeWM (Maes et al., 2026) and our Delta-JEPA (Zhang et al., 2026) re-implementation, two models whose encoder and predictor are trained jointly. We train both for 50 epochs on Push-T and also evaluate the released LeWM checkpoint.

Measuring ranking. The audit measures how well a scorer ranks a fixed set of candidate action sequences. The audit covers many anchors, and the i-th anchor $A_i \equiv (o_i, g_i, \mathcal{C}_i)$ is defined by an observation o_i, the goal observation g_i to be reached from o_i, and the set $\mathcal{C}_i$ of candidates to be ranked.

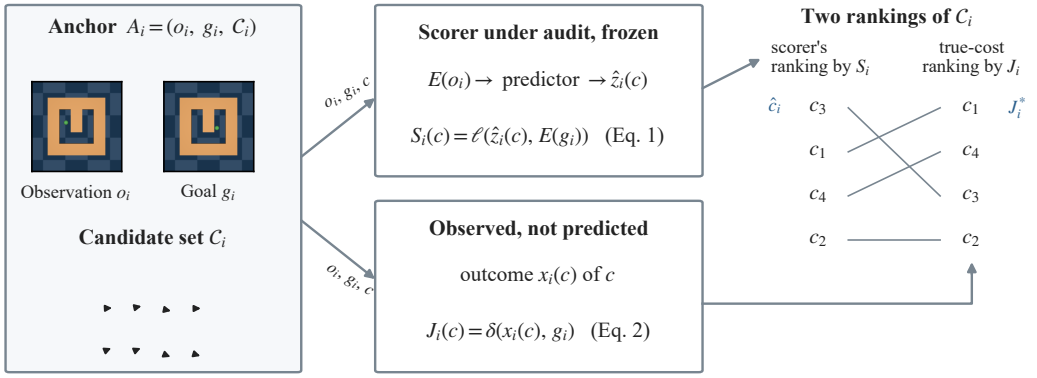


Figure 3: **How the audit scores and ranks one anchor's candidates.** The candidates are fixed before any scorer runs, and the scorer never receives an observed outcome $x_i(c)$ or a true cost $J_i(c)$. The audit measures the disagreement between the scorer's ranking and the true-cost ranking and reports Hit@1 (Equation 3), top-1 regret (Equation 4) and Spearman over the whole ranking. The two frames are a PointMaze observation and goal, and the two rankings of four candidates are an illustration, not data. The audit never calls the planner's search, so the search budget does not enter its results.

For each candidate $c \in \mathcal{C}_i$, write E for the encoder and $\hat{z}_i(c)$ for the predicted latent that the predictor reaches from $E(o_i)$ at the end of c. Let $S_i(c)$ be the planning objective's score of c in anchor A_i,

$$S_i(c) = \ell\big(\hat{z}_i(c),\, E(g_i)\big), \tag{1}$$

where ℓ measures how far the predicted latent lies from the goal latent $E(g_i)$, and Appendix A gives the exact form of ℓ on each task. A lower score ranks a candidate higher. Scores use the settings that Appendix A lists.

We judge the scorer's ranking against a correct ranking, which comes from the candidates' true costs, observed rather than predicted. Because each task measures distance from the goal differently, the true cost of candidate c in anchor A_i is

$$J_i(c) = \delta\big(x_i(c),\, g_i\big), \tag{2}$$

where $x_i(c)$ is the observed outcome of c and δ is the task's measure of how far that outcome falls short of the goal observation g_i. Appendix A gives the four definitions. However, the scorer never receives a true cost or a candidate's observed outcome. The scorer's ranking therefore comes from the model's predictions alone. Moreover, the audit (Figure 3) never calls the planner's search, so the search budget of Section 5 does not enter its results.

Five metrics follow, each comparing the scorer's ranking with the true-cost ranking within one anchor, then averaged over the anchors of an audit. Let $\mathcal{A}$ be the indices of the anchors in one audit, which are all anchors on PointMaze and Wall and, on Push-T and MetaWorld, those whose candidates do not all share one true cost (Appendix A). For $i \in \mathcal{A}$, let $\hat{c}_i = \arg\min_{c \in \mathcal{C}_i} S_i(c)$ be the top-scored candidate and $J_i^* = \min_{c \in \mathcal{C}_i} J_i(c)$ be the lowest true cost in anchor A_i. An anchor is misranked when $J_i(\hat{c}_i) > J_i^*$, and the wrong-anchor rate is the fraction of misranked anchors. Hit@1 is the fraction of anchors that are not misranked,

$$\text{Hit@1} = \big|\{\, i \in \mathcal{A} \mid J_i(\hat{c}_i) = J_i^* \,\}\big| \,/\, |\mathcal{A}|, \tag{3}$$

where $|\cdot|$ is the number of elements of a set, so Hit@1 and the wrong-anchor rate sum to one. An anchor whose top-scored candidate ties for the lowest true cost is therefore not misranked. A random ranking is scored by the same rule, so its Hit@1 exceeds one over the candidate count. Top-1 regret is what the wrong pick costs in the task's own units, averaged over the same anchors,

$$\text{Regret} = \frac{1}{|\mathcal{A}|} \sum_{i \in \mathcal{A}} \big(J_i(\hat{c}_i) - J_i^*\big), \tag{4}$$

where Regret is top-1 regret and an anchor that is not misranked adds zero. Hit@1 and the wrong-anchor rate count how many anchors are misranked, whereas top-1 regret measures how much the

wrong picks lose. A random ranking reaches a Hit@1 of 8.3% on PointMaze and 9.6% on Wall, where an anchor offers twelve candidates, and 5.8% on Push-T and 6.0% on MetaWorld, where it offers thirty-two. A wrong pick can be a near miss or a bad one, so on PointMaze and Wall we also report Hit@3, the fraction of anchors with a best candidate among the three top-scored. The Push-T and MetaWorld audits use a second form of Hit@3, so a Hit@3 value from one pair of tasks cannot be read against a value from the other. Finally, Spearman is the per-anchor rank correlation between score and true cost, positive when lower scores go with lower true costs and zero for a random ranking. Candidates come from released trajectories on PointMaze and Wall, and from scripted rollouts on Push-T and MetaWorld, so nothing guarantees a good candidate. The audit does not measure whether a planner generates a good candidate or succeeds in the closed loop; Section 5 takes up both.

Closed-loop runs. The released JEPA-WM checkpoints are reported to plan successfully, so we also run the planner to see what drives their success. We run PointMaze and Push-T, one navigation task and one manipulation task, with those checkpoints and JEPA-WM's CEM planner. PointMaze and Push-T are also the two tasks whose success rates Terver et al. (2025) find the least noisy. In our closed-loop runs, an episode lasts a fixed number of steps from an initial state, and the planner can replan as the episode proceeds. Because replanning can correct what a plan got wrong, planning less often should lower the success rate. Apart from the Introduction's search-budget comparison, we therefore vary only how often the planner replans: in one condition it replans after every action, and in the other it plans once at the start and that plan runs to the end of the episode. The two conditions share initial states and goals and differ in one variable, so they compare directly. In both conditions, CEM scores 320 candidates per plan, a reduced search budget against the 9,000 of JEPA-WM's released evaluation, so the success rates below are to be read against each other and not against published ones. An episode succeeds when its final state meets the task's success condition. Appendix A describes the four tasks, names the five encoders, and gives the second form of Hit@3 and the audit's remaining settings. Appendix D gives the training of LeWM and Delta-JEPA, and Appendix B the success conditions with the remaining planning settings.

4 Latent Distance Ranks Fixed Candidates Poorly

Closed-loop success measures the world model and the planner together, so it cannot say how well the model itself ranks. We therefore score candidates fixed before the model runs, not a set its own search has filtered.

4.1 Released models

The JEPA-WM checkpoints misrank most anchors on three of the four tasks (Figure 4 gives Hit@1 and top-1 regret relative to a random ranking's). On the two manipulation tasks, Push-T and MetaWorld, the planning objective's Hit@1 falls below what a random ranking reaches. On Push-T the planning objective's Hit@1 is 2.43%, against the 5.8% of a random ranking. On MetaWorld the planning objective never puts a best candidate first across all 750 anchors, a Hit@1 of 0.0% against a random ranking's 6.0%; its top-scored candidate holds the gripper still or acts at random on 745 of the 750 anchors (Appendix E).

By contrast, on the two navigation tasks the planning objective beats a random ranking's Hit@1 of 8.3% and 9.6%, reaching 31.6% on PointMaze and 50.8% on Wall over 256 anchors each. Under the transition the released Wall frames follow, Wall reaches 62.5% against a random 9.0% (Appendix A). On 32 anchors per task scored in one run (Appendix E), DINO-WM misranks most PointMaze anchors too but only 9 of the 32 Wall anchors. Table 1 gives the other metrics for each task. Scoring the observation a Push-T candidate actually reaches, in place of the predicted one, raises Hit@1 from 2.4% to 20.7%, so on Push-T the predictor causes the below-random Hit@1 (Appendix E).

4.2 Other encoders and end-to-end models

To examine whether poor action ranking persists across encoders (Figure 11, Appendix E), we compare DINOv2 and the four V-JEPA encoders on shared candidate sets in Push-T and MetaWorld reach-wall. All five models have a higher mean regret than a random ranking on Push-T, and four of five do on MetaWorld. V-JEPA 1 is the exception on MetaWorld: its regret is lower than a random

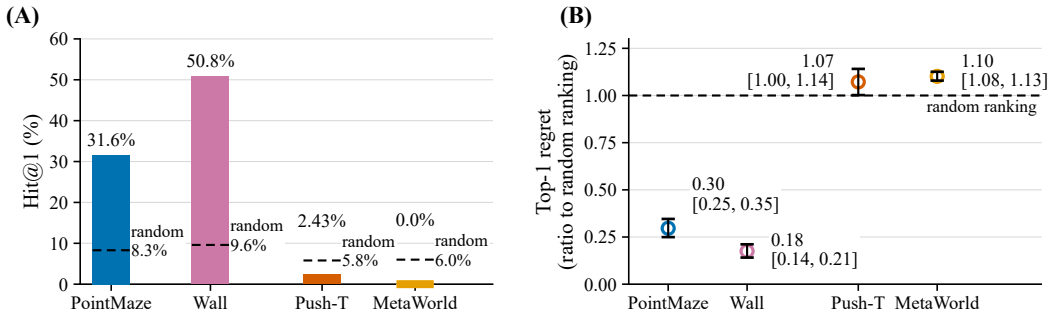


Figure 4: **The released checkpoints against a random ranking on fixed candidates.** Each task's released JEPA-WM checkpoint ranks fixed candidates under its own planning objective, with the objective weights Appendix A lists. (A) Hit@1, the fraction of anchors whose top-scored candidate has the lowest true cost, against the Hit@1 a random ranking reaches. MetaWorld's zero is marked by a stroke on the axis. (B) Top-1 regret, the true cost of the top-scored candidate above the lowest in its anchor, averaged over anchors and divided by a random ranking's top-1 regret on the same anchors, so one is the random ranking's level. PointMaze and Wall use 256 anchors each, Push-T and MetaWorld 618 and 750 of the 750 built, under the rule Appendix A gives. The planning objective beats a random ranking only on the two navigation tasks and not on the two manipulation tasks.

Task	Anchors	Hit@1	Wrong	Regret [95% CI]	Hit@3	Spearman
Twelve candidates per anchor						
PointMaze	256	31.6%	0.684	0.132 [0.11, 0.15]	0.629	0.581
Wall	256	50.8%	0.492	1.146 [0.92, 1.38]	0.809	0.623
Random (PointMaze)	—	8.3%	0.917	0.445 [0.42, 0.47]	—	0.000
Random (Wall)	—	9.6%	0.904	6.546 [6.25, 6.84]	—	0.000
Thirty-two candidates per anchor						
Push-T	618 / 750	2.43%	0.976	0.222 [0.21, 0.24]	0.165	0.065
MetaWorld	750 / 750	0.0%	1.000	0.283 [0.28, 0.29]	0.001	-0.597
Random (Push-T)	—	5.8%	0.942	0.207 [0.20, 0.22]	—	0.000
Random (MetaWorld)	—	6.0%	0.940	0.257 [0.26, 0.26]	—	0.000

Table 1: **Ranking of fixed candidates by each task's released checkpoint under its own planning objective, with the objective weights Appendix A lists and a random row per task.** Hit@1 and the wrong-anchor rate sum to one. The random rows give what a random ranking of the same candidates reaches. On Push-T and MetaWorld the two counts are the anchors retained and the anchors built, since an anchor whose candidates all share one true cost is dropped. Hit@3 takes a different form in each block, as Section 3 states, so values from the two blocks should not be read against each other. The released checkpoints rank no better than a random ranking on both manipulation tasks.

ranking's, but its top-scored candidate never has the lowest true cost. Figure 13 in Appendix E shows one misranked anchor.

For models trained end to end, we examine whether the latent distance favors actions from expert demonstrations. Every model is scored on the same 500 anchors, each starting from an observation in a Push-T expert trajectory, and we compare the recorded continuation, the actions the expert took next, with alternative action sequences, using the later observation from that trajectory as the goal. Among each anchor's 32 candidates, the recorded continuation has the lowest latent distance, including ties, at 69.6% of anchors for released LeWM, 85.0% for our 50-epoch LeWM, and 52.8% for our Delta-JEPA, against about 5% for a random ranking.

5 Where Planning Success Comes From

The released checkpoints rank fixed candidates poorly, as the previous section shows, and are nonetheless reported to succeed. This section measures what carries their success.

5.1 LESS REPLANNING LOWERS SUCCESS

At our reduced search budget, planning once rather than replanning lowers success from 89.6% to 61.8% on PointMaze and from 36.1% to 12.5% on Push-T. At the larger budget of the Introduction the same comparison on PointMaze gives 91.0% and 76.4% (Appendix B). The reduction appears on both tasks in all three seeds under which every run was repeated, and almost all episodes whose outcome changes between the two conditions change the same way. On PointMaze 48 episodes change, 44 succeeding only with replanning and 4 only without it; on Push-T 36 change, 35 only with replanning. The reduction is significant on both tasks under the exact paired test of Appendix G.

On Push-T, however, planning once with the same total search as replanning succeeds in 70 of the 144 episodes against 52 with replanning ($p = 0.0039$, Appendix F), so there the gain comes from the extra search. In sum, on the JEPA-WM checkpoints we measure the two parts of the ranking–success mismatch that Section 2 names: the checkpoints often misrank fixed candidates, and their closed-loop success depends on how often the planner replans or how widely it searches.

5.2 AN ORACLE SELECTION RULE RESTORES SUCCESS

Planning once may lower the success rate because the episode is too long for any plan to run without correction, or because the planner picks the wrong plan. In the last round of each search, the planner's own selection rule scores 64 candidates and executes the average of the eight top-scored, a new sequence for which no score is ever computed. Changing the selection rule cannot affect the first reason, since the environment sets how long an episode runs, but does affect the second, since the rule decides exactly which plan the planner executes. We therefore keep planning once and change only the selection rule. If the success rate rises back toward that of replanning, the planner was picking the wrong plan. We replace the selection rule with two others, each executing one of the 64 candidates rather than the average. The first draws one of those candidates with equal probability. The second is an oracle: it executes all 64 candidates from the current state in a separate simulator instance and then picks the one whose terminal state comes closest to meeting the task's success condition. The oracle is a diagnostic and not a rule a planner could use, because it needs the terminal state each candidate reaches, which a planner does not have when it plans.

Table 2 gives the three rules on both tasks, over 144 episodes each. On PointMaze the planner's own rule succeeds in 55.6% of episodes here against 61.8% in Section 5.1; Appendix C draws the rules' rates in Figure 6 and gives the reason for the difference. The three rules nonetheless run on the same initial states and goals, so their comparison is unaffected. An exact paired test finds the oracle significantly better and the random rule significantly worse than the planner's own rule, on PointMaze and Push-T alike. To see whether the oracle only picks out a good candidate that was already among the 64, we check each episode for a candidate that reaches the goal when executed. The last row of Table 2 counts the episodes holding such a candidate, which by construction are the episodes the oracle succeeds in. The oracle therefore bounds what a rule choosing among those 64 candidates could recover.

The oracle's 64 candidates hold a successful one in 132 of 144 PointMaze episodes, and on those same candidates the top-scored one succeeds in only 90, so in 42 episodes the model passes over a successful candidate it has already scored. On Push-T only 38 of the 144 episodes hold a successful candidate, and the top-scored one succeeds in 24 of them. Yet the planning objective puts the best of the planner's own candidates first in 18.1% of Push-T searches, against 1.6% at random (Appendix E).

6 DISCUSSION

Attempts to fix the ranking. Section 5.2 showed that PointMaze candidates usually hold a successful one, and that executing it raises the success rate. Unlike the oracle, a learned scorer needs no execution and can run inside the planner, so we trained two such scorers, one on PointMaze and one on Push-T. Both rank fixed candidates better than the planning objective, and neither raises the success rate (Appendix H, Figure 16). An improvement in ranking therefore counts only after being tested inside the planner. The two scorers failed for different reasons (Appendix H): the first ranks by more than the success condition checks, and none of the candidates the second ranked could succeed. On Push-T too the success rate falls when a scorer ranks by more than the success condition checks. Over the 144 episodes in which the planner plans once, a second oracle that ranks by the goal's full

	PointMaze		Push-T	
Selection rule	episodes	rate	episodes	rate
Random	48	33.3%	2	1.4%
Planner	80	55.6%	23	16.0%
Oracle	132	91.7%	38	26.4%
Episodes holding a good candidate	132		38	

Table 2: **Success under three selection rules, planning once, out of 144 episodes per task.** The rules differ in what they execute after scoring 64 candidates. Planner, the released planner's rule, executes the average of the eight top-scored candidates. The last row counts the episodes whose candidates include one that reaches the goal when executed. The oracle succeeds in exactly the episodes of the last row, and no rule that executes one of the scored candidates can do better.

state succeeds in 38.2% of PointMaze episodes and would succeed in 9.7% of Push-T episodes when rechosen from executed candidates. The planner's own rule reaches 55.6% and 16.0% on the same episodes, and the Section 5.2 oracle, which ranks by what the success condition checks, reaches 91.7% and 26.4%. What a scorer ranks by therefore can matter more than how accurately it ranks, because a perfect ranking by the wrong quantity does worse than the planner's own rule, which ranks poorly.

What to report when a world model plans. Current studies that plan with a world model judge the model by its closed-loop success rate. Few of them report a success rate without replanning (Zhou et al., 2024; Terver et al., 2025; Wang et al., 2026a). Sections 4 and 5 show that a success rate measured with replanning does not show whether the model picks the right action. We recommend reporting the success rate without replanning, and the ranking measured directly, alongside the success rate with replanning. A reader can then tell how much of the success comes from the model picking well and how much from replanning. Three limits bound the measurements behind this recommendation. We audit only simulated tasks (Appendix I), we ran the closed loop on PointMaze and Push-T alone, and both oracles are diagnostics rather than rules a planner could run.

7 Conclusion

Planning with a latent world model treats a candidate action sequence as good when the candidate's predicted latent lies close to the goal latent, and whether that closeness tracks the candidate's true cost across encoders has not been measured directly. We made that measurement on candidate sets fixed in advance, and the released checkpoints misrank most anchors on three of the four tasks and nearly half on Wall. The failure appears with image- and video-pretrained encoders alike, and on Push-T the predictor causes the below-random Hit@1. On the planner's own candidates the same scores rank far above random. The released checkpoints nonetheless succeed often. We ran the planner on PointMaze and Push-T to find out why, and on PointMaze replanning holds the success rate up, while on Push-T its gain comes from the extra search. In 132 of 144 PointMaze episodes the candidates of one planner search include a successful action sequence, yet the one the model scores highest succeeds in only 90. Choosing correctly among those candidates would therefore raise the success rate and leave the planner less dependent on frequent replanning. No scorer we tried makes that choice, and a scorer that measures what the success condition checks remains to be found.

AI use statement

Large language models were used to polish the manuscript and to search for related work. The authors reviewed and revised all text and take full responsibility for the content.

Reproducibility statement

The released checkpoints and the simulation environments are public. Appendices A to I give the audit and closed-loop settings and the statistical procedures with their seeds. The supplementary material holds the code, the anchor and candidate sets, and the per-candidate true costs and scores.

REFERENCES

Mahmoud Assran et al. V-JEPA 2: Self-Supervised Video Models Enable Understanding, Prediction and Planning, 2025. arXiv:2506.09985.

Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mahmoud Assran, and Nicolas Ballas. Revisiting Feature Prediction for Learning Visual Representations from Video, 2024. arXiv:2404.08471.

Y. Cheng, X. Ma, X. Wang, M. Yang, R. Su, H. Liu, X. Chen, S. Li, and Q. Li. AdaReP: Adaptive Re-Planning under Model Mismatch for Neural World-Model Predictive Control, 2026. arXiv: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, 2024. URL https://arxiv.org/abs/2303.04137.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2021. URL https://arxiv.org/abs/2004.07219.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning Latent Dynamics for Planning from Pixels. In *International Conference on Machine Learning*, 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.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering Diverse Domains through World Models, 2023. arXiv:2301.04104.

Jessica B. Hamrick, Abram L. Friesen, Feryal Behbahani, Arthur Guez, Fabio Viola, Sims Witherspoon, Thomas Anthony, Lars Buesing, Petar Veličković, and Théophane Weber. On the Role of Planning in Model-Based Deep Reinforcement Learning. In *International Conference on Learning Representations*, 2021. arXiv:2011.04021.

Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal Difference Learning for Model Predictive Control. In *International Conference on Machine Learning*, 2022.

Nicklas Hansen, Zhecheng Yuan, Yanjie Ze, Tongzhou Mu, Aravind Rajeswaran, Hao Su, Huazhe Xu, and Xiaolong Wang. On Pre-Training for Visuo-Motor Control: Revisiting a Learning-from-Scratch Baseline. In *Proceedings of the 40th International Conference on Machine Learning, Proceedings of Machine Learning Research vol. 202*, 2023. arXiv:2212.05749.

Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, Robust World Models for Continuous Control. In *International Conference on Learning Representations*, 2024.

Kei Honda, Ryo Yonetani, Masahiro Nishimura, and Tadashi Kozuno. When to Replan? An Adaptive Replanning Strategy for Autonomous Navigation using Deep Reinforcement Learning. In *IEEE International Conference on Robotics and Automation*, 2024.

Yingdong Hu, Renhao Wang, Li Erran Li, and Yang Gao. For Pre-Trained Vision Models in Motor Control, Not All Policy Learning Methods are Created Equal. In *Proceedings of the 40th International Conference on Machine Learning, Proceedings of Machine Learning Research vol. 202*, 2023. arXiv:2304.04591.

Nathan Lambert, Brandon Amos, Omry Yadan, and Roberto Calandra. Objective Mismatch in Model-based Reinforcement Learning. In *Proceedings of the 2nd Annual Conference on Learning for Dynamics and Control (L4DC), Proceedings of Machine Learning Research vol. 120*, pp. 1–16, 2020. arXiv:2002.04523.

Liangyu Li, Shengzhi Wang, Libin Qiu, Mingliang Xiong, and Qingwen Liu. World Model Control by Trajectory Reachability Metrics, 2026. arXiv:2605.22164.

Lucas Maes, Quentin Le Lidec, Damien Scieur, Yann LeCun, and Randall Balestriero. LeWorldModel: Stable End-to-End Joint-Embedding Predictive Architecture from Pixels, 2026. arXiv:2603.19312.

Lorenzo Mur-Labadia, Matthew Muckley, Amir Bar, Mahmoud Assran, Koustuv Sinha, Michael Rabbat, Yann LeCun, Nicolas Ballas, and Adrien Bardes. V-JEPA 2.1: Unlocking Dense Features in Video Self-Supervised Learning, 2026. arXiv:2603.14482.

Hoang Nguyen, Xiaohao Xu, and Xiaonan Huang. Latent Geometry Beyond Search: Amortizing Planning in World Models, 2026. arXiv:2605.08732.

Maxime Oquab et al. DINOv2: Learning Robust Visual Features without Supervision, 2023. arXiv:2304.07193.

Basile Terver, Tsung-Yen Yang, Jean Ponce, Adrien Bardes, and Yann LeCun. What Drives Success in Physical Planning with Joint-Embedding Predictive World Models?, 2025. arXiv:2512.24497.

Y. Wang, O. Bounou, G. Zhou, R. Balestriero, T. G. J. Rudner, Y. LeCun, and M. Ren. Temporal Straightening for Latent Planning, 2026a. arXiv:2603.12231.

Y. Wang, O. Bounou, Y. LeCun, and M. Ren. AdaJEPA: An Adaptive Latent World Model, 2026b. arXiv:2606.32026.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2021. URL https://arxiv.org/abs/1910.10897.

Zhenghao Zhang, Yuanxiang Wang, Zhenyu Guan, Yujia Yang, Bingkang Shi, Tianyu Zong, Hongzhu Yi, Guoqing Chao, Xingchen Chen, Tiankun Yang, Chenxi Bao, Tao Yu, Jingjing Zhou, and Jungang Xu. Delta-JEPA: Learning Action-Sensitive World Models via Latent Difference Decoding, 2026. arXiv:2606.31232.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. DINO-WM: World Models on Pre-trained Visual Features Enable Zero-shot Planning, 2024. arXiv:2411.04983.

A RANKING AUDIT DETAILS

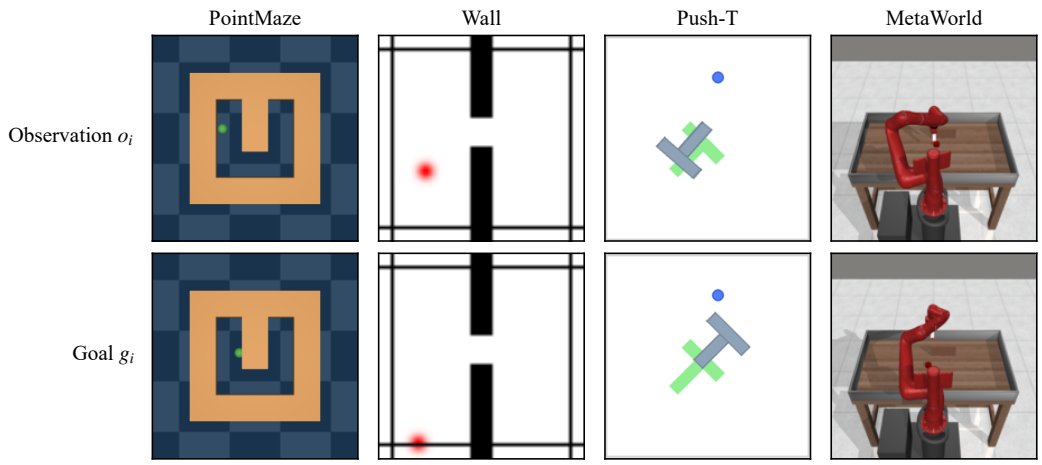


Figure 5: **One frame of an observation and of a goal from each of the four tasks.** PointMaze and Wall are navigation tasks and Push-T and MetaWorld reach-wall are manipulation tasks. Each task measures distance from the goal in a different quantity.

The four tasks are as follows. PointMaze comes from the D4RL benchmark suite (Fu et al., 2021) and asks an agent to apply a force that moves a ball through a U-shaped maze to a goal position. Wall comes from DINO-WM (Zhou et al., 2024) and asks an agent in one of two rooms to reach a goal in the other, through the one door in the wall between them. Push-T comes from the Diffusion Policy work (Chi et al., 2024) and asks an agent with a round pusher to push a T-shaped block to a target position and orientation. From MetaWorld (Yu et al., 2021), a suite of robot-arm tasks, we take reach-wall, which asks the arm to move its gripper to a goal position beyond a wall. As for the models, the encoders compared in Section 4.2 are the image-pretrained DINOv2 ViT-S/14 (Oquab et al., 2023) and the video-pretrained V-JEPA 1 ViT-L/16 (Bardes et al., 2024), V-JEPA 2 ViT-L/16 and ViT-g/16 (Assran et al., 2025), and V-JEPA 2.1 ViT-G/16 (Mur-Labadia et al., 2026). Each encoder is frozen and paired with a six-layer predictor of DINO-WM's architecture, trained per task without proprioceptive input and with a one-step rollout loss. Training runs 50 epochs at a constant learning rate of 5×10^{-4}, weight decay 10^{-7} and seed 234, on 224-pixel inputs for DINOv2 and 256-pixel inputs for the V-JEPA encoders, in batches of 16, or 8 for ViT-g and ViT-G. The ranking anchors are new simulator resets rather than frames from the training data, and each configuration is in the release. No validation set is used and no checkpoint is selected, so each audit scores the last checkpoint that training saves. As for the metrics, both top-1 regret and Hit@1 are measured against the best candidate in the same anchor, so neither depends on how good an anchor's candidates are. The candidates come from three sources, and how many an anchor offers depends on the task: twelve on PointMaze and Wall and thirty-two in Push-T and MetaWorld. On PointMaze and Wall each candidate is the action sequence that a released trajectory follows after the time step of the last frame of the anchor's observation, and the pool of released trajectories excludes the trajectories the anchors are drawn from. An anchor's twelve candidates come from the trajectories whose positions lie nearest to the anchor's at the time step of the last frame of its observation. In Push-T the candidates are rollouts from sixteen scripted push heuristics, two per heuristic, each one executed from a simulator reset with the seed that generated the anchor. In MetaWorld the candidates come in sixteen kinds, two of each. The first is the scripted expert's own action prefix, cut to the thirty actions a candidate holds. Seven more modify that prefix, adding Gaussian noise of scale 0.05, 0.15 or 0.35, reversing the order of its actions, negating its x, y and z components, or setting its first or its second half to zero. The other eight do not use the expert. They are no action, an action drawn at random at each step or drawn once and held, the largest action along each of the $+x$, $-x$, $+y$ and $-y$ directions, and a sinusoid on the x, y and z components. The expert rollout that defines the goal state runs longer than thirty actions, so the expert's prefix usually does not reach the goal, and the prefix still holds the anchor's lowest true cost on 69 of the 750 anchors. On every task the true cost comes from executing the candidate from the anchor's state. On Push-T and MetaWorld a true cost is computed once and

stored with the anchor before any scorer runs. On PointMaze and Wall the executions ran after the scores had been stored, and the scores were not recomputed. The true cost of Equation 2 is, on each task,

$$J_i(c) = \begin{cases} \left\| p(c) - p(g_i) \right\|_2 & \text{on PointMaze and Wall,} \\ 1 - \max_{1 \le t \le 30} r_t(c) & \text{on Push-T,} \\ \left\| s(c) - s(g_i) \right\|_2 & \text{on MetaWorld.} \end{cases} \tag{5}$$

Here, $p(c)$ and $p(g_i)$ are two-dimensional positions and $r_t(c)$ is the Push-T dense reward after the t-th of the candidate's thirty actions. Push-T's true cost therefore reads the candidate's best step, while the planning objective predicts its last. Scoring the same audit by $1 - r_{30}(c)$ instead retains 596 of the 750 anchors and gives a Hit@1 of 3.9% against a random ranking's 5.9%, so the planning objective falls below a random ranking under either definition. Here, $s(c)$ and $s(g_i)$ are MetaWorld simulator state vectors at the end of the candidate's execution and at the end of an expert rollout from the same start. On PointMaze and Wall, $p(c)$ is the position reached when the candidate's ten actions are executed from the anchor's state at the last frame of its observation o_i, a state that holds position and velocity on PointMaze, and $p(g_i)$ is the position in the anchor's own trajectory ten environment steps after that frame. The PointMaze and Wall goal observation g_i is the frame of the anchor's own trajectory in which $p(g_i)$ is measured. PointMaze is executed in the released MuJoCo maze model stepped with MuJoCo 2.1.2, and Wall with the transition of the released Wall data generator. Replayed from their stored states, both simulators reproduce every stored trajectory the audit uses, to within 10^{-5} in position on PointMaze and exactly on Wall. The released Wall frames follow a different rule, that of the evaluation environment, which moves the agent by twice the action, so after the first frame the frames the model sees and the recorded states disagree; our Wall true costs follow the recorded states. Replaying the evaluation environment's transition from each Wall trajectory's start reproduces 27,957 of the 28,050 stored frames of the audited trajectories to within half an intensity level, and the other 93 differ by at most 1.27 of 255. Under that transition each anchor's own actions reach the position its goal frame shows. The frames the model receives therefore follow that transition, while its proprioceptive input follows the recorded states. The Wall score is nonetheless almost entirely a distance between frame latents, because its proprioceptive term makes up about 0.002% of it, as reported below, although the proprioceptive input still enters the predictor. Table 1 reports the recorded states, and the evaluation environment's transition, which the frames follow, gives the values below. Under the evaluation environment's transition instead, executing each candidate from the position shown in the last frame of the anchor's observation gives a Hit@1 of 62.5% against a random ranking's 9.0%, a wrong-anchor rate of 0.375 and a top-1 regret of 1.039 [0.78, 1.32]. A random ranking's top-1 regret rises to 11.754 under that transition, because each action moves the agent twice as far. On the 32 anchors of Appendix E, DINO-WM then reaches a Hit@1 of 90.6% against JEPA-WM's 53.1%, with an exact paired p-value of 0.0005. For scoring, a candidate's actions are spliced onto the anchor's own observation and actions, so the score and the true cost describe the same start state and the same actions. The Push-T dense reward is the block's overlap with the target region, as a fraction of that region, divided by 0.95 and capped at one. The Push-T goal observation is a rendering of the scene with the block placed at its target pose. The MetaWorld goal observation is the observation at the end of the expert rollout that gives $s(g_i)$.

For the four JEPA-WM checkpoints, the encoder E of Section 3 maps an observation, an image with the proprioceptive state, to a latent $z \equiv (z^{\mathrm{v}}, z^{\mathrm{p}})$. Here, z^{v} holds the DINOv2 patch features of the image and z^{p} is an embedding of the proprioceptive state. The predictor advances the latent by one step for each group of five consecutive environment actions. On Push-T and MetaWorld the observation o_i is one frame, and the predictor takes six steps through the candidate's thirty actions. On PointMaze and Wall the observation o_i holds two frames five environment steps apart. The predictor there first receives the anchor's own five actions between those two frames and then takes two steps through the candidate's ten actions. The function ℓ of Equation 1 is

$$\ell(\hat{z}, z_g) = \frac{1}{d^{\mathrm{v}}} \left\| \hat{z}^{\mathrm{v}} - z_g^{\mathrm{v}} \right\|_2^2 + \lambda \, \frac{1}{d^{\mathrm{p}}} \left\| \hat{z}^{\mathrm{p}} - z_g^{\mathrm{p}} \right\|_2^2, \tag{6}$$

where $\hat{z} \equiv (\hat{z}^{\mathrm{v}}, \hat{z}^{\mathrm{p}})$ is the predicted latent at the end of the candidate and $z_g \equiv (z_g^{\mathrm{v}}, z_g^{\mathrm{p}})$ is the goal latent $E(g_i)$. Here, d^{v} and d^{p} are the numbers of entries in the two parts, and λ weights the proprioceptive part. The score is thus a mean of squared differences over the entries of each part. Only the predicted latent at the end of the candidate enters the score, as in the released configuration, which does not sum over the intermediate steps. On PointMaze and Wall the goal latent comes from the same encoder pass over the anchor's own frames that yields $E(o_i)$. Push-T and MetaWorld use

the released weight $\lambda = 0.1$. The audit's setting differs from the released configuration on PointMaze and Wall. The Table 1 scorer there uses $\lambda = 1$ against the released $\lambda = 0.1$, an unweighted sum set before the held-out split was created and not tuned, and the Appendix E scorer uses the visual latent alone. With $\lambda = 0.1$ the PointMaze Hit@1 is 30.9% rather than 31.6%, the top-1 regret 0.155 [0.13, 0.18] rather than 0.132 against a random 0.445, and the Spearman correlation 0.500 rather than 0.581. On Wall the proprioceptive term makes up about 0.002% of the score, so the Wall values of Table 1 are unchanged at their printed precision.

On PointMaze and Wall, Hit@3 is the fraction of anchors with a best candidate among the three top-scored, whereas on Push-T and MetaWorld it is the fraction of anchors whose top-scored candidate is among the three lowest in true cost. A random ranking meets the maze definition with a chance of three in twelve on PointMaze and 0.272 on Wall, where executed candidates tie at the lowest true cost on 11 of the 256 anchors. It meets the Push-T and MetaWorld definition with a chance of 0.212 and 0.113, since candidates tied at the third-lowest true cost all count. Ties at the lowest true cost count the same way, which is why the random Hit@1 of Section 3 is 5.8% and 6.0% rather than one in thirty-two. A random ranking drawn once on the PointMaze and Wall anchors reaches a Hit@1 of 10.5% and 8.6%. Neither of the two rates departs much from the 8.3% and 9.6% that the tie rule gives.

Algorithm 1 states the audit behind Table 1 in the order the scripts run it, and each line either names the equation that states the step or follows the code where the equations stop. On Push-T and MetaWorld, lines 3 to 6 compute the true cost of every candidate of an anchor before lines 7 to 13 score any of them; on PointMaze and Wall they ran after the scores were stored. In both orders the scorer receives the observation, the goal observation and the candidate's actions alone. Lines 15 to 27 then read the two stored tables and never call the scorer or the simulator, and the retained-anchor rule of line 23 is applied there, after every anchor has been scored and its per-anchor values computed. Beyond the symbols of Section 3, the algorithm writes n for the number of anchors built and P for the predictor, which also receives the earlier latents of its context window. For the predictor's rollout, $c^{(k)}$ is the k-th group of five consecutive actions of candidate c, H is the number of predictor steps, and z_k is the latent after k predictor steps, so that $z_0 = E(o_i)$. For the metrics, ε is the tolerance under which two true costs count as equal, $[\cdot]$ is the indicator of a condition, and lines 18 to 20 assign hit_i, regret_i and $\mathrm{hit3}_i$, anchor i's contributions to Hit@1, top-1 regret and Hit@3. Finally, $\hat{r}_i$ is one plus the number of candidates whose true cost lies strictly below $J_i(\hat{c}_i)$, r_i^* is the rank by score of the best-placed candidate whose true cost equals J_i^*, and ρ_i is the Spearman rank correlation between $S_i(c)$ and $J_i(c)$ over $c \in \mathcal{C}_i$. Marked with a triangle, the comments on the right name the equation a line applies or the task on which the line differs, and where the four tasks differ the comment states each task's value rather than adding a branch. The comments call PointMaze and Wall the mazes.

The protocol of the audits is stated in this appendix, and Appendices E and F give additional ranking and replanning results. Throughout the paper, each diagnostic that reads executed outcomes is labelled as a diagnostic where it appears, and none is presented as something a planner could run.

The PointMaze and Wall numbers we report come from a held-out split fixed in advance, which holds 256 anchors taken from released trajectories 640 to 895, an anchor per trajectory. The split's candidates come from released trajectories 896 to 1999 on PointMaze and 896 to 1919 on Wall. Candidates on Push-T and MetaWorld are scripted instead, so the contrast between the navigation and manipulation tasks is also a contrast between candidates drawn from data and scripted ones, which we do not separate.

The six checkpoints of Section 3 are files of the facebook/jepa-wms release: the four JEPA-WM files for PointMaze, Wall, Push-T and MetaWorld and the two DINO-WM files for PointMaze and Wall, each loaded with the released evaluation configuration for its task. We verified that the PointMaze and Wall JEPA-WM files are identical to a second download from the release, and that the Push-T and MetaWorld files match the copies in the release revision those two files were downloaded from. The two DINO-WM files were downloaded from the same release. Each checkpoint is loaded for inference only and without an optimiser, no gradients are computed while scoring, and no script writes to a checkpoint file. Where a later section trains a scorer or a head, the training touches that scorer or head alone and leaves the six checkpoints as released. Of the decoders a checkpoint holds, the Push-T and MetaWorld audits load only the image decoder, and the PointMaze and Wall audits load none. In all six checkpoints the image encoder is the publicly available DINOv2 ViT-S/14, which the release code loads in the same way for JEPA-WM and for DINO-WM. JEPA-WM and

Algorithm 1 Fixed-candidate ranking audit

Given anchors $A_i = (o_i, g_i, \mathcal{C}_i)$, $i = 1, \dots, n$, encoder E, predictor P, ℓ and δ
for $i = 1, \dots, n$ **do**
  for $c \in \mathcal{C}_i$ **do** ▷ the scorer never reads $x_i(c)$ or $J_i(c)$
    $x_i(c) \leftarrow$ outcome of c ▷ execute c from the anchor's state
    $J_i(c) \leftarrow \delta\big(x_i(c), g_i\big)$ ▷ Equation 5
  end for
  $z_0 \leftarrow E(o_i)$; $z_g \leftarrow E(g_i)$ ▷ mazes: one pass over the anchor's frames gives both
  for $c \in \mathcal{C}_i$ **do**
    for $k = 1, \dots, H$ **do** ▷ $H = 6$; mazes: $H = 2$, after the anchor's own first group
      $z_k \leftarrow P\big(z_{k-1}, c^{(k)}\big)$
    end for
    $\hat{z}_i(c) \leftarrow z_H$; $S_i(c) \leftarrow \ell\big(\hat{z}_i(c), z_g\big)$ ▷ Equations 1 and 6; last latent only
  end for
end for ▷ Push-T: lines 3–6 run for ten anchors before line 7; mazes: after all n were scored
for $i = 1, \dots, n$ **do** ▷ from the stored tables of J_i and S_i
  $\hat{c}_i \leftarrow \arg\min_{c \in \mathcal{C}_i} S_i(c)$ ▷ first row on a tied score
  $J_i^* \leftarrow \min_{c \in \mathcal{C}_i} J_i(c)$
  $\mathrm{hit}_i \leftarrow \big[\, J_i(\hat{c}_i) - J_i^* \le \varepsilon \,\big]$ ▷ Equation 3; $\varepsilon = 10^{-12}$, 10^{-9}, 0 on mazes, Push-T, MetaWorld
  $\mathrm{regret}_i \leftarrow J_i(\hat{c}_i) - J_i^*$ ▷ summand of Equation 4
  $\mathrm{hit3}_i \leftarrow [\, \hat{r}_i \le 3 \,]$ ▷ mazes: $[\, r_i^* \le 3 \,]$ instead
  $\rho_i \leftarrow$ Spearman of S_i against J_i over $\mathcal{C}_i$ ▷ undefined when all $J_i(c)$ or all $S_i(c)$ coincide
end for
$\mathcal{A} \leftarrow \{\, i : \max_c J_i(c) - \min_c J_i(c) > 10^{-6} \,\}$ ▷ Push-T, MetaWorld; mazes: all n anchors
Hit@1 $\leftarrow \frac{1}{|\mathcal{A}|} \sum_{i \in \mathcal{A}} \mathrm{hit}_i$; wrong-anchor rate $\leftarrow 1 -$ Hit@1 ▷ Equation 3
Regret $\leftarrow \frac{1}{|\mathcal{A}|} \sum_{i \in \mathcal{A}} \mathrm{regret}_i$ ▷ Equation 4; interval from 10,000 resamples of $\mathcal{A}$
Hit@3 $\leftarrow \frac{1}{|\mathcal{A}|} \sum_{i \in \mathcal{A}} \mathrm{hit3}_i$; Spearman $\leftarrow$ mean of the defined ρ_i over $i \in \mathcal{A}$
return Hit@1, wrong-anchor rate, Regret, Hit@3, Spearman ▷ Table 1

DINO-WM therefore differ after the encoder, in the predictor and in whether the action enters that predictor as a token or as a feature.

B PLANNING DETAILS

In the released configurations every environment is observed as an RGB image of 224 by 224 pixels together with the simulator's proprioceptive state, and Wall draws its image at 65 by 65 pixels before resizing it. On PointMaze the action is a two-dimensional force on the ball, with each component in the range $[-1, 1]$. On Wall the action is a two-dimensional displacement of the agent, each component in the same range and scaled by two, and a move that meets the wall stops there unless the move passes through the door. The Wall layout is fixed, with one wall position and one door position, and the start and goal of an episode lie on opposite sides of the wall. In the copy of the Push-T environment that JEPA-WM ships, the action is a two-dimensional target position for the pusher, relative to its current position. The fixed-candidate Push-T audit executes its candidates in the gym-pusht package's implementation of the same environment, whereas the closed-loop runs use JEPA-WM's copy. On MetaWorld the task is the reach-wall variant on a simulated Sawyer arm, and the task's state vector includes the goal position. The action has four components in $[-1, 1]$: a three-dimensional displacement of the gripper and a gripper command. MetaWorld itself counts reach-wall as solved when the centre of the gripper lies within 0.05 of the goal position, whereas JEPA-WM's evaluation counts success when the full 39-dimensional state lies within 0.3 of the goal state, and we use JEPA-WM's condition throughout. A MetaWorld episode has 100 steps.

The closed-loop runs use CEM with a planning horizon of six planner actions (each spanning five environment steps), five rounds, 64 samples per round and eight elites, the top-scored samples kept at each round. The plan executed after each search is the mean of the Gaussian that CEM fits to the elites of its last round, not one of the sampled candidates. JEPA-WM's released evaluation runs 30 rounds with 300 samples and ten elites at the same horizon, so CEM scores 9,000 candidates per

plan there against 320 here. Each task runs 48 episodes under each of three seeds, 70001 to 70003 on PointMaze and 71001 to 71003 on Push-T, giving 144 episodes per task under each condition, whereas the released evaluation runs 96 episodes under seed 1.

A closed-loop episode runs 30 environment steps, which the planner covers in six planner actions, and the two replanning intervals are one planner action and six planner actions. On PointMaze the start and goal of an episode are drawn with a per-episode seed that depends only on one of the three seeds above and on the episode index. On Push-T a random-number generator that advances only with the episode index picks a segment of the released Push-T dataset, and the start and goal come from that segment, so the same index gives the same episode under both intervals. On PointMaze an episode succeeds when its final position lies within 0.5 of the goal position. On Push-T an episode succeeds when the four-dimensional vector of agent and T-shaped block positions lies within 20 of the goal's and the block angle lies within $\pi/9$ of the goal angle. Both success conditions are applied to the simulator state at the last step of the episode only, so an episode that reaches the goal and then leaves the goal counts as a failure.

Only the intervals one and six have three seeds. The run over the intervals one, two, four and six covered a single seed of 48 episodes per task, under the same budget. The selection runs of Section 5.2 keep the planner, budget, seeds and episode count above, fix the interval at six, and run their three selection rules on one environment instance per seed, with every rule seeing the same episodes.

Settings and uncertainty for Figure 1. Panel (a) compares two candidates from the last CEM round of the same search on PointMaze. The top-scored candidate has a latent distance of 0.614 and misses the goal, whereas the candidate that reaches the goal has a latent distance of 0.627, with lower distances preferred. In panels (b) and (c), the smaller budget is our reduced search budget, with 64 samples per round, five rounds and eight elites, and the larger budget is that of the released evaluation, with 300 samples per round, 30 rounds and ten elites. Panel (b) compares these budgets on Push-T with one plan executed over 30 environment steps. Panel (c) uses the larger budget on PointMaze and compares one plan with replanning every five environment steps. Each comparison uses 144 paired episodes across three seeds. Push-T success is 12.5% under the smaller budget and 67.4% under the larger budget, with episode-level 95% Wilson intervals of [8.1, 18.9] and [59.3, 74.5] percent. PointMaze success is 76.4% with one plan and 91.0% with replanning, with corresponding intervals of [68.8, 82.6] and [85.2, 94.6] percent.

Illustration in Figure 2. The PointMaze scenes come from a simulator replay. The scenes show the initial observation and the state after five environment steps; the goal observation shown is rendered from the episode's recorded goal state. The action symbols are schematic. Within the search loop, CEM updates its sampling distribution using the mean and standard deviation of the top-scored action sequences.

C SELECTION EXPERIMENT DETAILS

The three selection rules of Section 5.2 run one after another under the settings of Appendix B. With the replanning interval fixed at six planner actions, an episode holds one planning decision, so each rule makes one choice per episode. The planner's own rule executes the mean that CEM fits after its fifth round, and with the released momentum setting of zero that mean is the average of that round's eight top-scored candidates. The other two rules choose among the 64 candidates of the fifth round, captured before CEM fits that mean, so every candidate they can choose was scored. Each rule runs its own search, and because the random draws continue from one rule to the next, each rule draws its own 64 candidates, so the three rules share an initial state but not a sample.

The oracle is a diagnostic that executes each candidate from the episode's initial state in a separate simulator instance, without rendering, and reads the resulting simulator state. On PointMaze the oracle picks the candidate whose final position is nearest the goal position, and the success condition compares that same distance with 0.5. On Push-T the oracle gives each candidate a value that measures how near the candidate comes to passing the success condition: the larger of the position difference divided by 20 and the angle difference divided by $\pi/9$. The oracle picks the candidate with the smallest value. A candidate's value is below one exactly when the candidate passes the condition.

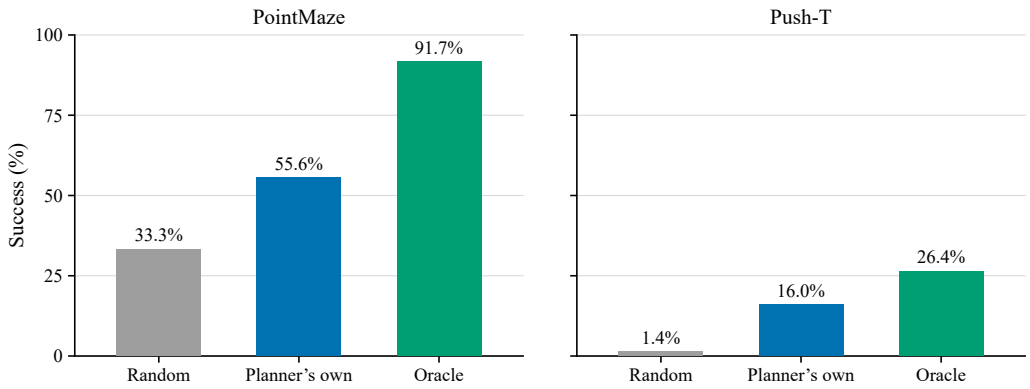


Figure 6: **Success over 144 episodes per task under three selection rules when the planner plans once and the selection rule changes.** The rules are a draw at random from the 64 scored candidates, the planner's own rule, which executes the average of the eight top-scored candidates, and the oracle, which executes all 64 candidates and picks the one that comes closest to meeting the success condition. The oracle is a diagnostic and not a rule a planner could run; it succeeds in exactly the episodes whose candidates include one that reaches the goal. Picking the right candidate would restore much of the lost success.

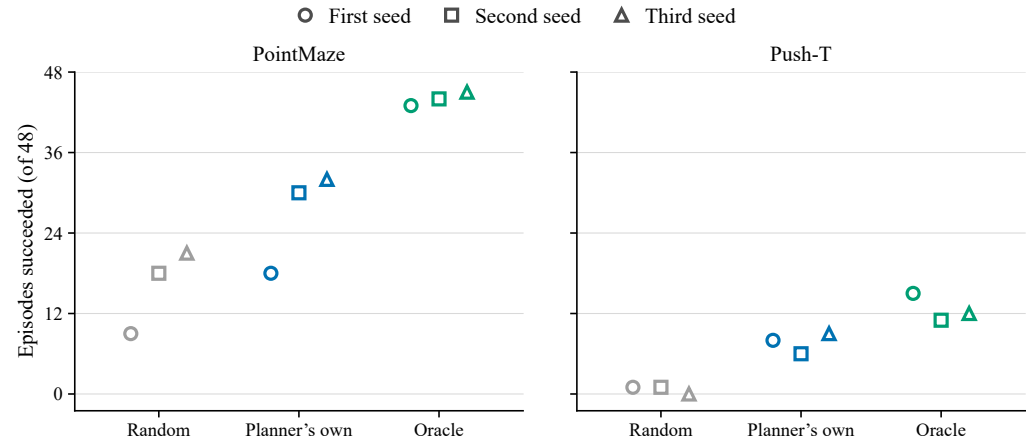


Figure 7: **Successful episodes, out of 48 per seed, under the three selection rules of Section 5.2, planning once.** Seeds are offset horizontally so that equal counts stay visible. The oracle exceeds the planner's own rule in all three seeds.

Per seed, the planner's own rule succeeds in 18, 30 and 32 of 48 PointMaze episodes, the oracle in 43, 44 and 45, and the random rule in 9, 18 and 21. On Push-T the planner's own rule succeeds in 8, 6 and 9 of 48, the oracle in 15, 11 and 12, and the random rule in 1, 1 and 0. Pooled over seeds, the oracle gains 52 PointMaze episodes against the planner's own rule and loses none, with a p-value from the exact paired test below 10^{-15}, and on Push-T the oracle gains 18 against 3 lost, with a p-value of 0.0015. The random rule gains 10 PointMaze episodes and loses 42, with a p-value of 9.1×10^{-6}, and on Push-T it gains none and loses 21, with a p-value of 9.5×10^{-7}. The oracle exceeds the planner's own rule in every seed on both tasks (Figure 7), although on Push-T only the first seed reaches significance on its own, with a p-value of 0.039 against 0.125 and 0.375, so the significance of the oracle's Push-T gain rests on the pooled test.

Among the oracle's 64 candidates, a mean of 23.6 succeed per episode on PointMaze and 1.06 on Push-T, where the median is zero. Executing the single top-scored candidate instead of the CEM mean that the planner's own rule executes would have succeeded in 90 of the 144 PointMaze episodes

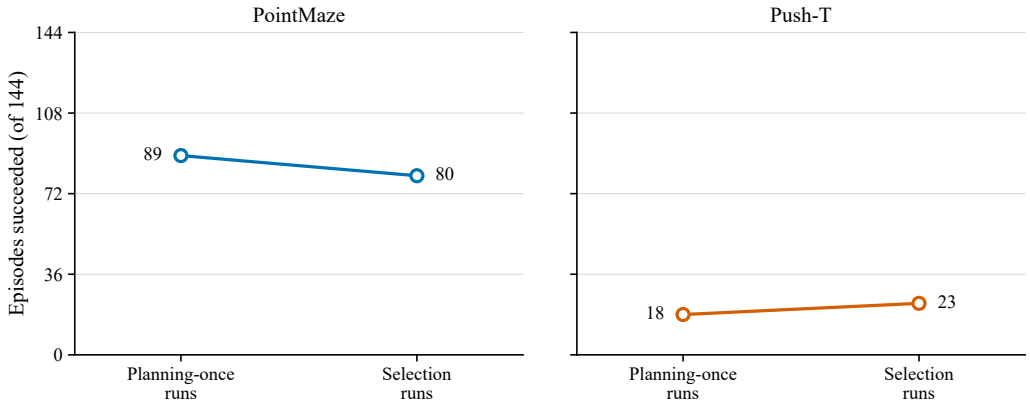


Figure 8: **Successful episodes, out of 144, under the planner's own rule in the planning-once runs of Section 5.1 and in the selection runs of Section 5.2, on the same seeds and episodes.** In the selection runs every rule draws from one random-number generator seeded once per run, so from the second episode on the planner's own rule draws different candidates. The two sets of runs differ in the candidates drawn, not in the rule.

and 24 of the 144 Push-T episodes, judged by the same simulator executions. The oracle's gain therefore comes from more than executing a scored candidate. On the planner's own candidates the top-scored one nonetheless beats a random pick by a wide margin, on Push-T unlike on the audit's fixed candidates.

The PointMaze selection runs were made in two campaigns, and the first campaign also ran a second diagnostic, an oracle on the environment's full-state evaluation distance, which includes the ball's velocity and a randomly drawn goal velocity. The full-state oracle succeeded in 55 of 144 episodes, significantly below the planner's own rule, with 45 episodes lost against 20 gained and a p-value of 0.0026. On Push-T the same ranking was not run separately; over the success oracle's executed candidates, choosing the smallest state distance would have succeeded in 14 of the 144 episodes. The two campaigns drew identical candidates at all 432 planning decisions, and the planner's own rule and the random rule gave exactly the same outcomes in both campaigns. The planner's own rule succeeds here in 80 of 144 PointMaze episodes and 23 of 144 Push-T episodes, against 89 and 18 in the planning-once runs of Section 5.1, which used the same seeds and episodes (Figure 8). The difference arises because here each episode runs one search per rule, and all three rules draw from a random-number generator seeded once per run, so from the second episode on, the planner's own rule draws different candidates than it did in those runs.

D LeWM and Delta-JEPA

For the LeWM and Delta-JEPA models we trained, the encoder is a randomly initialized ViT-Tiny with 14×14 patches and 192-dimensional latent vectors. Both models use a six-layer Transformer predictor with 16 attention heads, 64 dimensions per head, and a feed-forward width of 2,048. The predictor uses causal attention and incorporates action embeddings through adaptive layer normalization. We use 224×224 images and a context of three observations spaced five environment steps apart. The five two-dimensional actions between successive observations are concatenated into a 10-dimensional action input.

Delta-JEPA reconstructs these actions with a three-layer non-causal Transformer decoder containing five learned queries, one per environment step. The decoder has eight attention heads with 64 dimensions per head and a feed-forward width of 512. The difference between successive latent vectors is projected to 512 dimensions and conditions every decoder block through adaptive layer normalization.

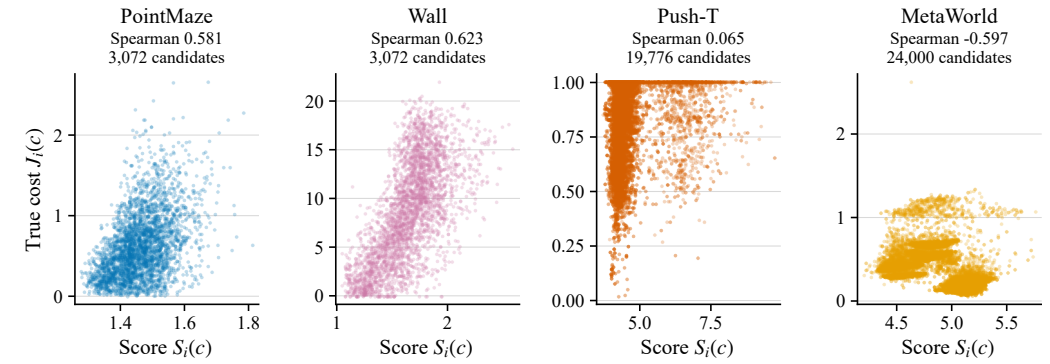


Figure 9: **Planning-objective score against true cost, one point per candidate of the released-checkpoint audit.** The axes are the score $S_i(c)$ of Equation 1 and the true cost $J_i(c)$ of Equation 2, in each task's own units, so every panel has its own axes and the panels are not compared with one another. Each panel states the mean per-anchor Spearman of Table 1 and the number of candidates drawn. Points are drawn with transparency, so darker regions hold more candidates. Score and true cost are nearly unrelated on Push-T and move in opposite directions on MetaWorld.

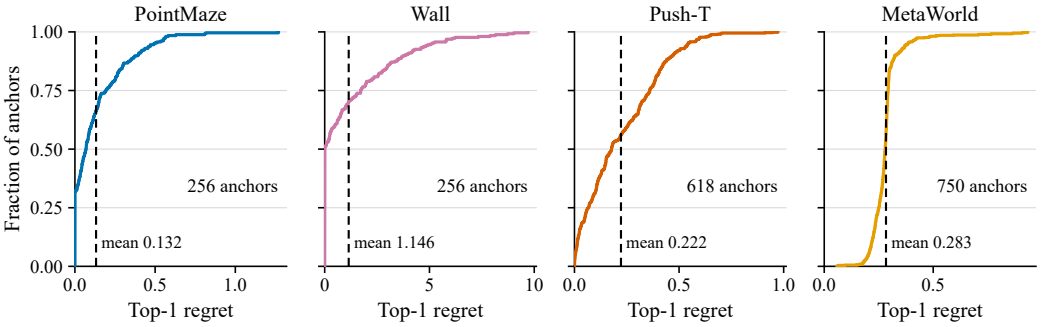


Figure 10: **How the per-anchor top-1 regret of Equation 4 is spread over the anchors of the released-checkpoint audit.** The dashed line marks the mean top-1 regret that the released-checkpoint table reports, and the jump at zero is Hit@1. The regret is spread unevenly across anchors.

Both models minimize a prediction loss $\mathcal{L}_{\text{pred}}$, the mean squared error between predicted and encoded latent vectors at the next model step. LeWM uses $\mathcal{L}_{\text{pred}} + 0.09\mathcal{L}_{\text{SIGReg}}$, where $\mathcal{L}_{\text{SIGReg}}$ encourages an isotropic Gaussian distribution of latent vectors. Delta-JEPA uses $\mathcal{L}_{\text{pred}} + 10\mathcal{L}_{\text{action}}$, where $\mathcal{L}_{\text{action}}$ is the mean squared error between the reconstructed and normalized recorded action inputs. The encoder and predictor are optimized jointly in both models, together with the action decoder in Delta-JEPA.

We use LeWM's Push-T expert dataset with a random 90%/10% training–validation split, generated with seed 3072. Images are normalized using ImageNet statistics. Each action coordinate is standardized using its mean and standard deviation over the full source dataset, computed before the split.

Both models are trained for 50 epochs using AdamW, a batch size of 128, a learning rate of 5×10^{-5}, and weight decay of 10^{-3}. We use the training framework's linear-warmup and cosine-annealing scheduler, updated once per epoch. Training uses bfloat16 precision and a gradient-clipping threshold of 1.0. Evaluation uses the epoch-50 checkpoints.



Figure 11: **Top-1 regret on Push-T and MetaWorld for the five encoders of the encoder-swap comparison of Section 4.2.** This audit generates its own anchors, so its Push-T retention differs from Table 1's: the bars cover 587 retained of 750 on Push-T and 750 on MetaWorld, with 95% bootstrap intervals, and the number beside each bar is its regret. The dashed line is the regret of a random ranking of the thirty-two candidates. On Push-T every encoder has a higher regret than a random ranking, and on MetaWorld every encoder except V-JEPA 1 does.

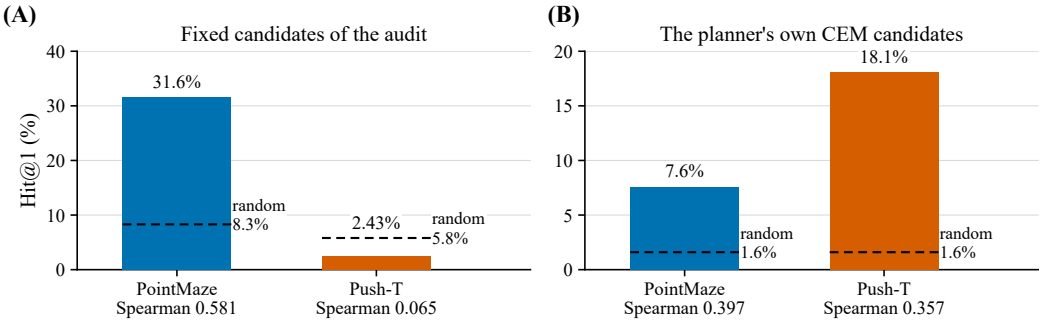


Figure 12: **Hit@1 of the planning objective on the audit's fixed candidates and on the planner's own candidates, on PointMaze and Push-T.** (A) The fixed candidates of Table 1. (B) The 64 candidates of the last search round in each of the 144 planning-once episodes of Section 5.2, scored against the quantity the oracle of Appendix C uses. The panels differ in candidate set, candidate count and true-cost definition, so each bar is read against its own dashed random level; ties at the lowest true cost count as correct, and the mean Spearman is printed under each task. Wall and MetaWorld were not run in closed loop. On Push-T the objective ranks the fixed candidates below a random ranking and its own candidates well above it; on PointMaze it is above on both.

E Additional ranking results

The planning objective ranks the planner's own Push-T candidates far better than it ranks the audit's. On the 144 planning-once episodes of Section 5.2, scoring the last round's 64 candidates by the objective and against the true cost each rule's simulator records gives a Hit@1 of 7.6% on PointMaze and 18.1% on Push-T, against 1.6% for a random ranking of sixty-four, with per-population Spearman of 0.397 and 0.357. On Push-T the audit's fixed candidates give 2.43% against a random 5.8% and a Spearman of 0.065. Figure 12 sets the two candidate sets side by side, each bar against its own random ranking. The objective therefore carries information about the candidates a planner produces, and the failure this paper measures is a failure on candidate sets fixed in advance; what Section 5.2 adds is that even among its own candidates the top-scored one succeeds in only 90 of the 132 PointMaze episodes that hold a successful one.

In the encoder-swap comparison a Push-T candidate's true cost is one minus its highest dense reward, as in Table 1, and a MetaWorld candidate's is the Euclidean distance between the environment's observation vector at the end of the candidate and at the end of the expert rollout that defines the goal. On MetaWorld the released checkpoint's top-scored candidate holds the gripper still on 491 of

the 750 anchors and acts at random on 254 more. The anchor's best candidate is instead the largest action along $+y$ on 619 anchors and the expert's prefix, alone or with noise added, on 129 of the others, and the checkpoint picks neither on any anchor. The negative Spearman of Table 1 therefore comes from a preference between kinds of candidate, not from a fine-grained misordering. A random ranking of the same thirty-two candidates puts the recorded continuation first, ties included, at 5.3% of anchors for released LeWM, 5.8% for our 50-epoch LeWM and 4.8% for our Delta-JEPA; the three differ because a tie is a tie in the model's own scores, so each model induces its own tie structure on the shared candidates. Figure 9 plots every candidate's score against its true cost on the four tasks of Table 1. An earlier version of the PointMaze audit scored a candidate by the distance averaged along its predicted rollout, not by the distance at the rollout's end that the released configuration uses. Scoring by the averaged distance picks a different candidate on 39% of the held-out anchors and gives a mean top-1 regret of 0.133 rather than 0.132 and a Hit@1 of 30.9% rather than 31.6%.

The Push-T row of the released-checkpoint table comes from a re-run of the full 750-anchor audit, made because the earlier run of this audit recorded the absolute pixel target of each action where the checkpoint's preprocessor expects a signed displacement from the pusher's current position. Only the recorded action changed, so the anchors, the seeds, the sixteen scripted push heuristics, the true costs and the scorer are the ones the earlier run used, and 618 of the 750 anchors are retained in both. Those 618 anchors hold 454 distinct scenarios, and Appendix G repeats the tests with each scenario counted once.

Changing the recorded action also changes what the scorer prefers, from the heuristic that draws each action independently at random, preferred on 510 of the 618 retained anchors, to a spread over the constant-action and sinusoidal heuristics. Relative action values outside the range the checkpoint saw in training make up 2.61% of the total, and 86% of those come from that same random heuristic. We left them unclipped so that the candidate set matches the earlier run exactly.

The DINO-WM comparison uses the 32-anchor protocol under which all four PointMaze and Wall checkpoints were scored in one run, and not the 256-anchor held-out split behind Table 1. Each candidate of those 32 anchors is executed from its anchor's full state, as in Table 1. On those anchors DINO-WM reaches a Hit@1 of 21.9% against JEPA-WM's 28.1% on PointMaze, and 71.9% against 40.6% on Wall. The exact paired test on the paired outcomes returns a p-value of 0.69 on PointMaze and 0.0020 on Wall, where DINO-WM alone is right on 10 anchors and JEPA-WM alone on none. One anchor is worth 3.1 points at this size, so the two models are separated on Wall and not on PointMaze.

The planning objective mixes the geometry of the latent distance with the predictor's error. To separate the two, we also score each candidate with the same planning objective applied to the encoded observation that the candidate actually reaches, with the goal unchanged. On Push-T this observed score reaches a Hit@1 of 20.7% against the planning objective's 2.4% and a random ranking's 5.8%, over the same 618 anchors. Its top-1 regret is 0.124 [0.113, 0.135], against 0.222 [0.208, 0.237] for the planning objective and 0.207 for a random ranking. The two scores pick the same top-scored candidate on 7.6% of anchors. Counting each repeated scenario once leaves 454 scenarios, on which the observed score reaches 19.4% and the planning objective 3.1%. On Push-T the latent distance thus ranks above random once the predictor is bypassed, although the observed score still misranks 79.3% of anchors, so the predictor is what puts Hit@1 below random. We do not report the measurement on PointMaze or Wall. On PointMaze our renders of a reached state differ from the stored frames by more than a tolerance fixed in advance, and on Wall the stored frames and states follow different transition rules. On MetaWorld we took the measurement on 100 of the 750 anchors, drawn at random with a seed fixed before any result was read. On those anchors the observed score reaches a Hit@1 of 55.0% against the planning objective's 0.0% and a random ranking's 5.9%, and a top-1 regret of 0.041 [0.023, 0.065] against 0.276 [0.266, 0.285] and a random 0.258. On MetaWorld the predictor thus accounts for most of the ranking failure, since bypassing it removes most of the top-1 regret. When an anchor's candidates are scored against another anchor's goal, the observed score's Hit@1 changes little, to 58% on MetaWorld and 18.3% on Push-T. On Push-T every goal shows the block at the same target pose, and on MetaWorld the best candidate is the largest action along $+y$ on 619 of the 750 anchors, so on both tasks a score can rank these candidates well without using the goal. The observed score's high Hit@1 therefore shows that the predictor loses ranking information, not that the latent distance tracks a particular goal.

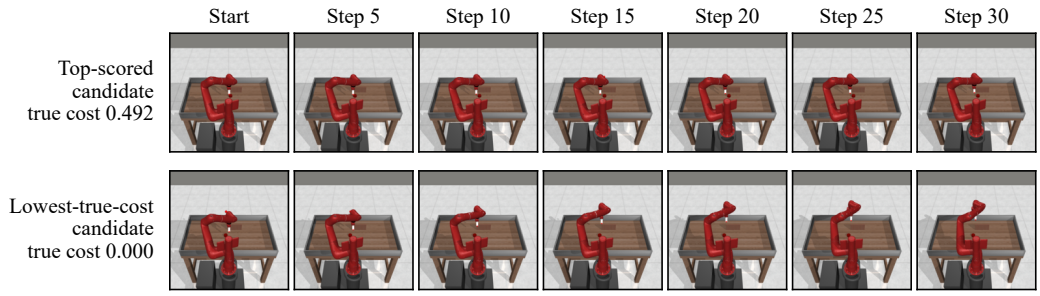


Figure 13: **Frames every five environment steps from executing two candidates of one anchor, for the V-JEPA 1 model of the encoder-swap comparison of Section 4.2 on MetaWorld reach-wall.** The model is not a released checkpoint. The top row executes the candidate the model's planning objective scores best, and the bottom row the lowest-true-cost candidate, which is the prefix of the scripted expert's own actions. In the encoder-swap comparison a candidate's true cost is a distance between observations, not the distance between simulator states used elsewhere in the paper, so these true costs are not comparable with those of Table 1. The anchor shown has the largest top-1 regret of the 750, at 0.492. On 743 of the 750 anchors the model's top-scored candidate applies no action.

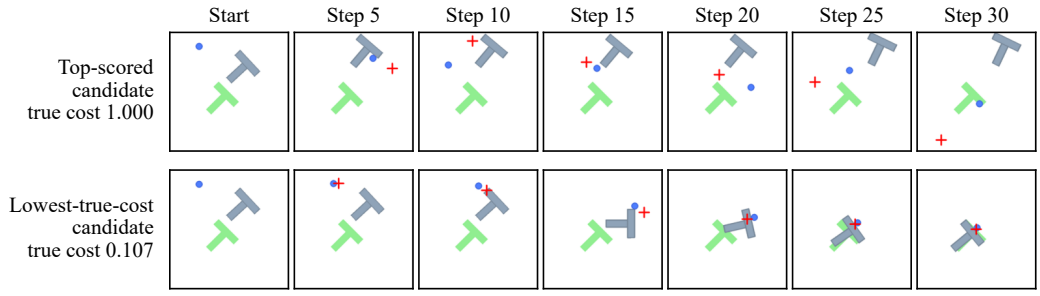


Figure 14: **Frames every five environment steps from executing two candidates of one anchor, for the V-JEPA 2 model of the encoder-swap comparison of Section 4.2 on Push-T.** The model is not a released checkpoint. Of the anchor's thirty-two scripted candidates, the top row executes the one the model's planning objective scores best and the bottom row the one with the lowest true cost. A candidate's true cost is one minus its highest dense reward. The grey T is the block, the blue disc the pusher, the green T the target pose, and the red cross the pusher's start. The top-scored candidate pushes the block away from the target; the lowest-true-cost candidate pushes the block onto the target.

Figures 13 and 14 show one misranked anchor frame by frame for two models of the encoder-swap comparison of Section 4.2, on MetaWorld reach-wall and on Push-T.

F Additional replanning results

On PointMaze the three seeds give success rates of 85.4, 91.7 and 91.7% with replanning, against 43.8, 68.8 and 72.9% without it. On Push-T the three seeds give 45.8, 29.2 and 33.3% against 10.4, 12.5 and 14.6%. When the seeds are pooled, the paired difference is 27.8 points on PointMaze with a 95% bootstrap interval of 19.4 to 36.1 points, and 23.6 points on Push-T with an interval of 16.7 to 30.6 points. The two-sided p-value from the exact paired test is below 10^{-8} on both tasks. Replanning after every planner action also runs the search six times per episode where planning once runs it once, so the two conditions differ in total search as well as in feedback. On PointMaze the extra search does not explain the difference: replanning at the reduced budget scores at most 1,920 candidates per episode and succeeds in 89.6% of episodes, more than planning once at the larger budget of 9,000 candidates, which succeeds in 76.4%. On Push-T planning once at the larger budget succeeds in 67.4%, more than replanning at the reduced budget, so that comparison cannot tell the two causes apart. We therefore also run Push-T planning once with 384 rather than 64 candidates per CEM

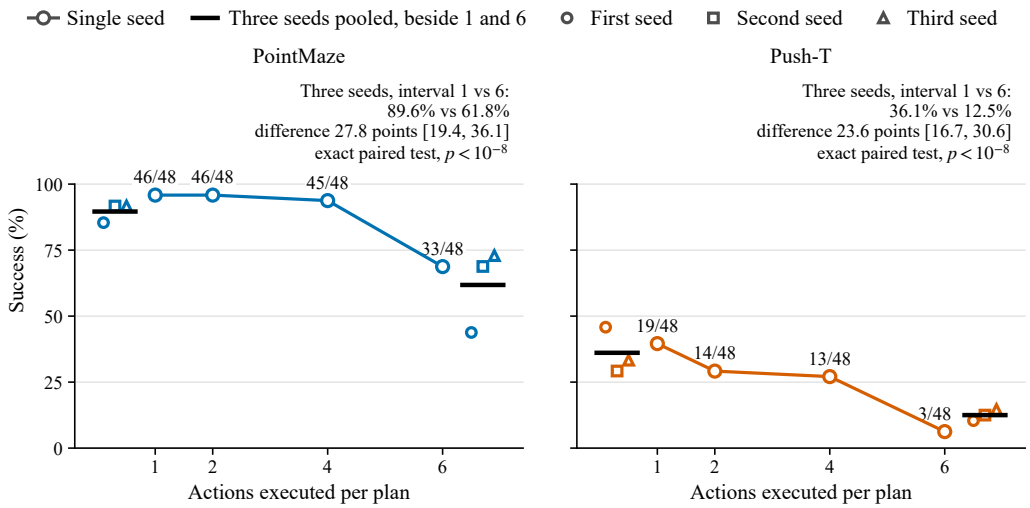


Figure 15: **Success rate against the number of actions executed per plan, from one to six, on PointMaze and Push-T, at a single seed and pooled over three.** The line gives the single-seed rate over 48 episodes, each point labelled with its count of the 48. Beside the points at one and six actions, the black bar gives the rate pooled over three seeds of 48 episodes each and the hollow markers give each seed's rate, with the seeds listed in Appendix B. Colour gives the task and marker shape gives the seed. Success falls as the planner replans less often, on both tasks and in all three seeds.

round, so that the single plan scores 1,920 candidates, the most that replanning scores in an episode, on the same 144 episodes. Planning once with that search succeeds in 70 of the 144 episodes (48.6%), against 52 (36.1%) with replanning and 18 (12.5%) when planning once at the reduced budget. This condition succeeds more often than replanning in all three seeds, and 27 episodes succeed only under this condition against 9 only with replanning (exact paired test, p = 0.0039). On Push-T, spending the same search on a single plan therefore matches and exceeds the gain from replanning at this budget, so there we find no benefit from the new observations that replanning uses.

On each task a separate run varies the replanning interval over one, two, four and six actions, at a single seed of 48 episodes (Figure 15). On PointMaze the planner succeeds in 46, 46, 45 and 33 of those 48 episodes across the four intervals, and on Push-T in 19, 14, 13 and 3. Those counts differ from the three-seed rates above because the sweep uses its own seed, 68000 on PointMaze and 69021 on Push-T, rather than one of the three, so we report the counts for the direction of the trend and not for its size.

G Statistical tests

Unless otherwise stated, the two scorers in every comparison are measured on the same anchors and the same fixed candidate sets. Figure 10 gives the distribution of the per-anchor regret behind each mean. Apart from the episode-level intervals quoted for Figure 1, which are 95% Wilson intervals, every confidence interval in the ranking audits and the closed-loop runs is a 95% percentile bootstrap, over anchors at 10,000 resamples for the audits and over paired episodes at 20,000 for the runs. Whether a checkpoint's Hit@1 falls below a random ranking's is tested one-sided against the exact distribution of the number of anchors a random ranking gets right, in which each anchor counts with the probability that a random pick ties for its lowest true cost. The probability of a count as low as the checkpoint's is 3.3×10^{-5} on Push-T, with 15 of 618 anchors right against 36.0 expected, and 7.3×10^{-21} on MetaWorld, with 0 of 750 against 44.9. On Push-T the top-1 regret, whose interval overlaps a random ranking's, cannot be told apart from it, so the below-random finding there rests on Hit@1. On each task 200 of the 750 anchors repeat an earlier anchor's seed and true costs, which leaves 454 distinct scenarios among Push-T's 618 retained anchors and 550 on MetaWorld, and the encoder-swap audit's MetaWorld anchors repeat in the same way. Counting each scenario once, the tail probability is 3.6×10^{-3} on Push-T, with a Hit@1 of 3.1% against 5.9%, and 1.7×10^{-15} on

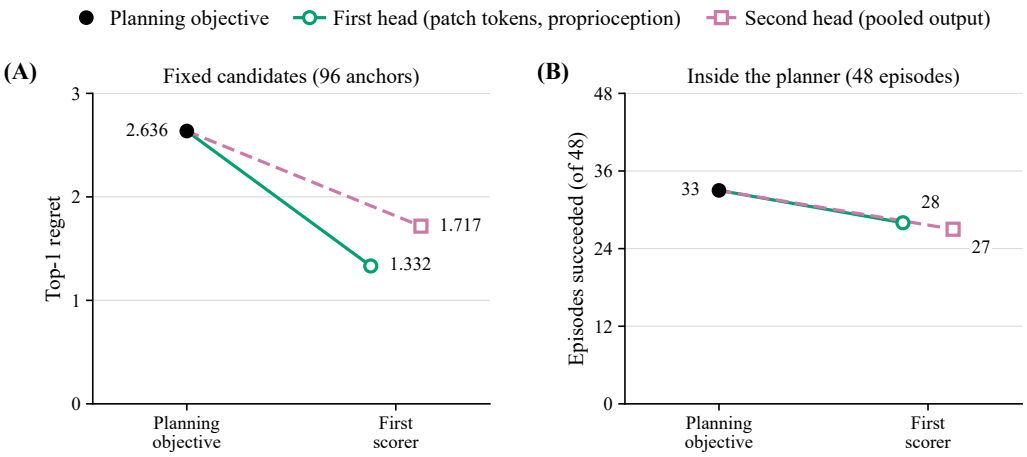


Figure 16: **The first scorer against the planning objective, on fixed candidates and inside the planner.** (A) Top-1 regret on 96 development anchors with 64 candidates each, for the planning objective and for the first scorer's two heads. (B) Episodes that succeed, out of 48, under the planner's own selection rule and when the planner's candidates are reranked with each head. The second head's line is dashed to stay visible where the two lines nearly coincide. Better ranking on fixed candidates does not carry into the planner.

MetaWorld, with 0 of 550 against 33.0 expected. The top-1 regret is then 0.219 [0.201, 0.236] against a random 0.206 on Push-T and 0.281 [0.275, 0.288] against 0.257 on MetaWorld. Paired binary outcomes are compared with the exact paired test, McNemar's test on the discordant pairs, whether the outcomes are two conditions' episode successes or two scorers' top-1 correctness. Every result the paper calls significant survives a Holm correction for as many as twenty tests, more than the paper reports; the single-seed Push-T test of Appendix C, at 0.039, does not, and that appendix already rests the Push-T oracle result on the pooled test.

H ATTEMPTS TO FIX THE RANKING

The two learned scorers of Section 6 are described here: how they were built, the candidates and episodes on which they were measured, and the per-condition numbers the main text leaves out.

For the first scorer, our re-implementation of the reachability cost of Li et al. (2026), we trained two heads over the frozen encoder's outputs, each an ensemble of three seeds. One head reads the encoder's patch features and the proprioceptive state separately before combining them, and the other reads the encoder's pooled output. Both heads were trained with a Smooth-L1 loss on how many frames apart two frames of one released trajectory are. We measured the first scorer twice, once on a set of candidates fixed in advance and once inside the planner (Figure 16). The fixed set holds 96 development anchors with 64 candidates each. On that set a candidate's true cost is replaced by its full-state cost, the distance from its terminal state to the goal's full state, velocity included. On that set the top-1 regret falls from 2.636 for the planning objective to 1.332 for the first head and 1.717 for the second. The first head's improvement has a 95% interval of [0.970, 1.633], and when the 96 anchors are split into six groups the first head has a lower regret than the planning objective in all six groups. Inside the planner the first scorer was run on 48 development anchors, one episode each, one plan per episode, with five CEM rounds of 64 candidates and eight elites. The planner's own selection rule succeeds in 33 of those episodes. Reranking with the first head succeeds in 28 and with the second head in 27. Using the first head as the search objective succeeds in 21, and a shuffled-label control succeeds in 5. Over the first 24 anchors the planner's own selection rule succeeds in 17 and reranking with the first head in 13. In 28 episodes across the two first-head variants, the planner's own selection rule succeeds and the first scorer fails. In 25 of those 28, once velocity is counted, the plan the first scorer picked ends closer to the goal than the one the planner's own selection rule picked, and in position alone it ends no closer.

On Push-T the second scorer predicts from the frozen encoder's outputs how much of the environment's fixed target region each candidate will cover. The second scorer's runs judge success differently from the rest of the paper. An episode counts as a success when the T-shaped block covers 95% of the fixed target region at any executed step. The second scorer was trained on 48 anchors with 64 candidates each. On those anchors no candidate reaches the coverage that counts as a success at any step, even when every candidate is executed first and the best is kept. Executing every candidate and keeping the best yields a rise in the mean highest dense reward from 0.229 under the planner's own selection rule to 0.357. That rise is the most the second scorer could recover. The second scorer recovers 51% of the rise from 0.236 under the planning objective's own top candidate on held-out data, and 48% of the corresponding rise on a further 48 anchors. A third set of 48 anchors was executed once with the second scorer and once without it. On that set the highest dense reward rises by 0.068 with a confidence interval of [0.027, 0.113] when the planner plans once, and by 0.028 with a confidence interval of [0.002, 0.055] when the planner replans after every action. In both cases the success count is 0 of 48 with the second scorer and 0 of 48 without it.

I SCOPE OF THE RECOMMENDATION

The real-robot dataset in JEPA-WM's release has no simulator in which to compute a true cost, so the audits of this paper cover the four simulated tasks alone. A world model on a real robot also faces sources of error that simulation lacks. Assran et al. (2025) report one such source, a camera placement that leaves the direction of an action ill-defined for the model. Whether the ranking failure persists on a real robot, and how large it is there, remains untested. A scorer that picks the right plan would also save planner calls. In our setting, replanning after every action calls the planner six times per episode and planning once calls the planner once, so the planner is called 83% less often. The saving would be large, because planning with V-JEPA 2-AC on a real robot takes about 16 seconds per action (Assran et al., 2025).