LEARNING LATENT WAYPOINTS FOR WORLD-MODEL PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Latent world models enable efficient planning by evaluating candidate actions in compact representation spaces, but terminal goal-matching costs provide little information about useful intermediate states. We propose Latent Residual Waypoints, a lightweight method for generating intermediate waypoints between start and goal latents. Given a frozen, pretrained world-model encoder, our method predicts encoded intermediate frames by learning a residual correction to latent linear interpolation. During planning, we generate a sequence of latent waypoints recursively. These are incorporated through a minimum-over-time waypoint cost that encourages trajectories to pass near future waypoints while still optimizing for the final goal. This enables effective planning in difficult settings, such as planning with budget constraints. A planner solving episodes of the Two-Rooms and Cube tasks under a constrained CEM budget can outperform full-budget final-goal planning. At matched budgets, the performance deltas are even more pronounced. We also find that our method improves Le-WorldModel's CEM planning performance across navigation and manipulation tasks and outperforms simpler waypoint methods like linear interpolation and nearest-neighbor snapping. We extend this to other world models and show improvements on gradient-based closed-loop MPC planning. Though our waypoints generally improve mean success rate compared to final-goal planning, they yield fewer gains on some easier tasks. Qualitative analysis of visualized latent waypoints and nearest-neighbor analyses further show that Latent Residual Waypoints remain closer to realistic trajectory latents than those produced by naive interpolation.

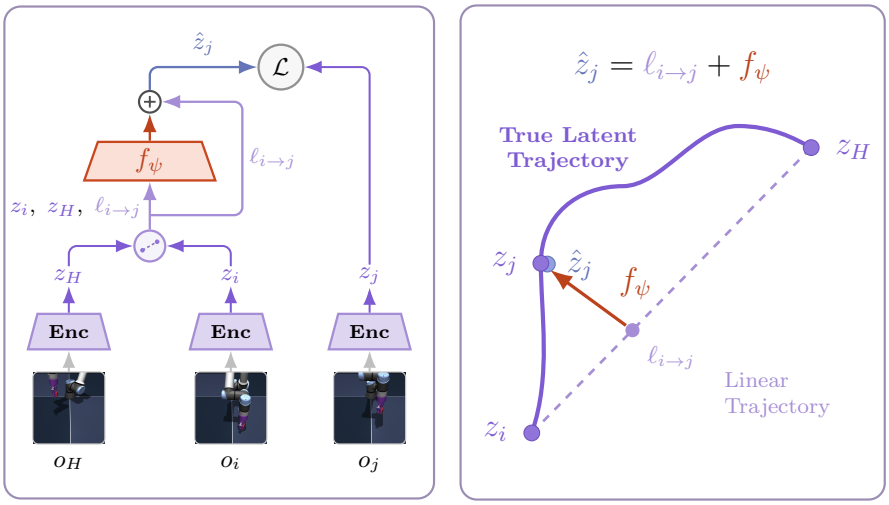


Figure 1: During training, start and end frames o_i and o_H and an intermediate frame o_j are encoded into latent states z_i, z_H, and z_j using a frozen world model encoder. To predict z_j, we first compute the linear step between z_i and z_H and then model the residual with f_ψ. The predicted residual is added to the linear step to produce the prediction $\hat{z}_j$.

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.

1 INTRODUCTION

Latent world models (Assran et al., 2025; Sobal et al., 2025; Zhou et al., 2024; Maes et al., 2026; Zhang et al., 2026) learn representations that support effective planning with simple terminal goal-matching costs. This is especially appealing for reconstruction-free models such as JEPA world models (LeCun, 2022), where candidate action sequences can be evaluated entirely in representation space by rolling learned dynamics forward, without pixel prediction. In this setting, goals are target embeddings, so the geometry of the latent space directly shapes the planner's optimization landscape.

Terminal goal matching is often effective, but goals may provide little signal about useful intermediate states. Prior work on temporal straightening (Wang et al., 2026) shows that locally straight latent trajectories can improve latent planning, suggesting that paths in latent space, not just individual latent embeddings, can implicitly encode information that provides planning guidance. However, trajectories in pretrained latent spaces are often not straight, and latent linear interpolation between start and goal may veer off-manifold or produce dynamically implausible intermediates. We therefore ask whether learned latent spaces can still provide useful intermediate waypoints that guide planners toward task solutions.

We propose Latent Residual Waypoints (Figure 1), a lightweight model that learns a residual correction to linear interpolation between start and goal latents and, at evaluation time, places waypoints along the resulting trajectory. These waypoints are then used to augment the planning objective through a minimum-over-time cost that encourages plans to pass through future waypoints while progressing toward the goal. The model is trained on trajectories in a pretrained latent space and is initialized from linear latent interpolation, learning only residual corrections toward true latent trajectories. This approach works directly with a pretrained world model without retraining.

We evaluate on a diverse set of manipulation and navigation tasks and consider three different pretrained world model encoders. Our experiments show that:

- **Latent Residual Waypoints enable planning in difficult scenarios.** Our method consistently improves performance under limited planning budgets. In the hardest setting with a constrained budget of CEM samples and optimization steps, it even surpasses final-goal planning using the full budget. On harder manipulation splits requiring larger object displacements, it also delivers meaningful performance gains for both CEM and Adam planning.
- **Latent Residual Waypoints enable more effective planning.** Our method improves LeWorldModel's CEM planning over terminal latent goal-matching and simple waypoint generation baselines, yielding improvements on navigation and manipulation tasks, including a substantial success gain in the hardest environment. We also find that in this setting it performs comparably to oracle-trajectory waypoint embeddings. We show that these improvements extend to other world model encoders, improving the performance of DINO-WM and a ResNet world model on navigation tasks. Although our method generally outperforms final-goal planning, it also yields lower gains on some easier tasks, tying or losing to some simpler waypoint methods.
- **Latent Residual Waypoints are plausible intermediate latents.** When decoding linear interpolation waypoints to images, we find that they lose coherence in the middle of the trajectory. Our latent waypoints, by contrast, decode to visually coherent frames and appear to track the oracle trajectory. Quantitatively, they are also closer to both embedded oracle waypoints and embedded dataset frames than waypoints produced by linear interpolation.

2 RELATED WORK

Planning with Waypoints or Subgoals: Prior work used intermediate waypoints or subgoals to improve planning. In classical planning, landmarks are components required by any valid plan (Porteous et al., 2001; Porteous & Cresswell, 2002) and can be used to define subgoals and improve performance (Karpas & Domshlak, 2009; Hoffmann et al., 2004). More contemporary methods use replay-buffer observations as waypoints (Eysenbach et al., 2019), generate subgoals as images or Chain-of-Thought-like intermediates (Lippi et al., 2020; Nair & Finn, 2020; Ni et al., 2024; Kang & Kuo, 2025), or recursively produce latent subgoals in a coarse-to-fine manner (Liu et al., 2025; Huang et al., 2024). Most closely related are methods that generate sequences of latent goals (Xie

et al., 2025; Liu et al., 2025). We avoid diffusion-based planning by training a residual model to produce a single cheap deterministic rollout.

Latent Space Planning: Our work relates to latent-space planning, where learned latent dynamics support effective planning across diverse environments (Hafner et al., 2019a;b; Schrittwieser et al., 2020). Embed to Control (Watter et al., 2015) shares an intuition with our method, but embeds into a latent space with locally linear dynamics, whereas we learn a residual correction to a linear latent trajectory. Related work encourages temporally straight representations for planning (Wang et al., 2026) or learns contrastive representations for generating intermediate waypoints (Eysenbach et al., 2024).

Trajectory Optimization: Because our method learns a correction from an initial linear latent trajectory, it also relates to prior work on trajectory optimization. Research in classical planning primarily involved producing trajectories that avoid collisions, for example through graph methods (Kavraki et al., 1996). Additional refinements to improve trajectory quality, such as removing hard stops, can be provided through optimization with artificial forces (Quinlan & Khatib, 1993; Brock & Khatib, 2002) and smoothness priors (Zucker et al., 2013). Contemporary work has optimized trajectories directly with a diffusion model (Carvalho et al., 2023) or via gradient descent in a learned latent space (Srinivas et al., 2018).

3 Background

Latent World Models: Latent world models learn to predict environment dynamics in latent space rather than directly in observation space. An encoder $\mathrm{enc}(\cdot)$ maps each observation o_t to a latent state

$$z_t = \mathrm{enc}(o_t).$$

Depending on the architecture, z_t may be represented as a single vector, as in Le-WorldModel (Maes et al., 2026) or retain spatial structure, as in DINO-WM (Zhou et al., 2024) and the ResNet world model of Wang et al. (2026).

To model action-conditioned dynamics, an action encoder $\mathrm{enc}_a(\cdot)$ maps actions into representations that condition a predictor $\mathrm{pred}(\cdot)$. Given a history of latent states and actions of length L, the predictor estimates the next latent state z_{t+1}:

$$\hat{z}_{t+1} = \mathrm{pred}(z_{t-L+1:t}, \mathrm{enc}_a(a_{t-L+1:t}))\,.$$

Different architectures vary in how action information is introduced into the predictor. For example, DINO-WM concatenates action representations with the latent features whereas Le-WorldModel uses them to modulate layer normalization.

Linear interpolation: Let $z_0, z_H \in \mathbb{R}^d$ denote two endpoint latents, where z_0 is the start latent and z_H is the goal latent. Linear interpolation defines the straight-line path between them using a progress parameter $\alpha \in [0, 1]$, $z(\alpha) = (1-\alpha)z_0 + \alpha z_H = z_0 + \alpha(z_H - z_0)$. To generate K evenly spaced waypoints along this segment, including the endpoints, we choose $\alpha_k = \frac{k}{K-1}$ for $k = 0, 1, \ldots, K-1$ and define evenly spaced points $z_k = z(\alpha_k) = z_0 + \frac{k}{K-1}(z_H - z_0)$.

This can also be written as an incremental step. Suppose the current latent z_i lies on the straight-line path at progress α_i, i.e., $z_i = z(\alpha_i)$. For a later progress value $\alpha_j > \alpha_i$, the corresponding straight-line latent $z(\alpha_j)$ can be expressed directly in terms of z_i and the goal latent z_H. Since $z_H - z_i = (1-\alpha_i)(z_H - z_0)$, we obtain

$$z(\alpha_j) = z_i + \frac{\alpha_j - \alpha_i}{1 - \alpha_i}(z_H - z_i). \tag{1}$$

4 Latent Residual Waypoints

Latent Residuals: Let o_i, o_j, and o_H denote observations from a trajectory at steps i, j, and horizon step H, with $i < j \leq H$. We define normalized trajectory progress by $\gamma_t := \frac{t}{H}$.

Each observation is embedded into latent space using a frozen, pretrained encoder, giving $z_i = \mathrm{enc}(o_i)$, $z_j = \mathrm{enc}(o_j)$, $z_H = \mathrm{enc}(o_H)$. The goal is to learn the latent trajectory flow from the current latent z_i to a future latent z_j, conditioned on the trajectory endpoint latent z_H. We first define a straight-line latent step from z_i toward z_H at the relative progress level γ_j:

$$\ell_{i\to j} := z_i + \frac{\gamma_j - \gamma_i}{1 - \gamma_i}\,(z_H - z_i)\,. \tag{2}$$

The denominator is clamped when γ_i is close to 1. The model learns a residual correction around this linear baseline. We provide our predictor f_ψ with the current latent, endpoint latent, linear baseline latent, and progress scalars $x_{i,j} = [z_i,\, z_H,\, \ell_{i\to j},\, \gamma_i,\, \gamma_j,\, \gamma_j - \gamma_i]$ and train it to model the residual from the linear step to the true future latent:

$$\hat{z}_j = \ell_{i\to j} + f_\psi(x_{i,j}), \tag{3}$$

where f_ψ is parameterized by a trainable MLP.

Training: During training we sample trajectory triplets (o_i, o_j, o_H), embed them with the frozen encoder, and optimize the latent residual parameters ψ. The primary objective is mean squared error in latent space, $\mathcal{L}_{\mathrm{mse}} = \frac{1}{d}\,\|\hat{z}_j - z_j\|_2^2$, but we also include a cosine distance term to encourage the predicted latent to align with the true latent, $\mathcal{L}_{\cos} = 1 - \frac{\hat{z}_j^\top z_j}{\|\hat{z}_j\|_2 \|z_j\|_2}$.

We combine these terms in the total objective $\mathcal{L}$:

$$\mathcal{L} = \mathcal{L}_{\mathrm{mse}} + \lambda \mathcal{L}_{\cos} \tag{4}$$

Waypoint Generation: At evaluation time, the model generates intermediate latent waypoints for each start-goal pair. Let $z_{\mathrm{start}} = \mathrm{enc}(o_{\mathrm{start}})$ and $z_{\mathrm{goal}} = \mathrm{enc}(o_{\mathrm{goal}})$. Given a desired number of waypoints K, including the start and goal, we produce a sequence of uniformly spaced progress points $0 = \alpha_0 < \alpha_1 < \cdots < \alpha_{K-2} < \alpha_{K-1} = 1$, $\alpha_i = \frac{i}{K-1}$.

We initialize $\tilde{z}_0 = z_{\mathrm{start}}$ and then recursively produce new waypoints toward the fixed goal using the predicted latents:

$$\tilde{z}_m = \ell_{m-1\to m} + f_\psi(x_{m-1,m}), \qquad m = 1, \ldots, K-2. \tag{5}$$

giving a trajectory $(z_{\mathrm{start}}, \tilde{z}_1, \ldots, \tilde{z}_{K-2}, z_{\mathrm{goal}})$, with endpoints set by the true start and goal latents.

Planning with latent waypoints: At evaluation time, planning evaluates candidate action sequences using the frozen world model. For a candidate plan, let $\hat{z}_{1:H}$ denote the predicted latent rollout over the planning horizon. The terminal goal cost is

$$\mathcal{C}_{\mathrm{goal}}(\hat{z}_{1:H}) = \|\hat{z}_H - z_{\mathrm{goal}}\|_2^2\,. \tag{6}$$

We aim to augment this cost by encouraging the agent to pass near waypoints that lie ahead of it along the generated path. Let $\tilde{z}_0 = z_{\mathrm{start}}$ and $\tilde{z}_{K-1} = z_{\mathrm{goal}}$, and let $z_{\mathrm{cur}} = \mathrm{enc}(o_{\mathrm{cur}})$ denote the current observation embedding. We interpret the sequence of waypoints $(\tilde{z}_0, \ldots, \tilde{z}_{K-1})$ as a piecewise-linear path. For each segment $j \in \{0, \ldots, K-2\}$, define the straight line between consecutive waypoints as $v_j = \tilde{z}_{j+1} - \tilde{z}_j$ and compute

$$\beta_j = \mathrm{clip}_{[0,1]}\left(\frac{\langle z_{\mathrm{cur}} - \tilde{z}_j, v_j\rangle}{\max(\|v_j\|_2^2, \epsilon_d^2)}\right), \tag{7}$$

where $\epsilon_d > 0$ ensures numerical stability. β_j gives the fraction of the current latent along the segment. We can use this to compute the projected point $\pi_j = \tilde{z}_j + \beta_j v_j$ and select the closest such projected point, $j^\star = \arg\min_j \|z_{\mathrm{cur}} - \pi_j\|_2^2$. Writing the cumulative path length at waypoint i as

$$s_0 = 0, \qquad s_i = \sum_{j=0}^{i-1} \|v_j\|_2, \qquad s_{\mathrm{cur}} = s_{j^\star} + \beta_{j^\star}\|v_{j^\star}\|_2, \tag{8}$$

we define the set of intermediate waypoints ahead of the agent by

$$\mathcal{A}(z_{\mathrm{cur}}) = \{i \in \{1, \ldots, K-2\} \;:\; s_i > s_{\mathrm{cur}} + \epsilon_s\}\,, \tag{9}$$

where $\epsilon_s > 0$ is a small progress tolerance. The final goal is excluded from this auxiliary set because it is already represented by the terminal goal cost.

We assign each intermediate waypoint a weight based on its Euclidean distance from the current latent state:

$$q_i = \mathbf{1}[i \in \mathcal{A}(z_{\text{cur}})] \left(\|\tilde{z}_i - z_{\text{cur}}\|_2 + \epsilon_d\right)^{-p}, \qquad \omega_i = \frac{q_i}{\max\left(\sum_{k=1}^{K-2} q_k, \epsilon_d\right)}. \tag{10}$$

Here $p = 0$ gives uniform weighting over the remaining waypoints, while $p > 0$ places greater weight on nearby waypoints.

The **minimum-over-time** waypoint cost is then given as:

$$\mathcal{C}_{\text{ahead}}(\hat{z}_{1:H}; z_{\text{cur}}) = \sum_{i=1}^{K-2} \omega_i \min_{t \in \{1,\dots,H\}} \|\hat{z}_t - \tilde{z}_i\|_2^2, \tag{11}$$

Yielding total cost:

$$\mathcal{C}(\hat{z}_{1:H}; z_{\text{cur}}) = \mathcal{C}_{\text{goal}}(\hat{z}_{1:H}) + \lambda_{\text{wp}} \mathcal{C}_{\text{ahead}}(\hat{z}_{1:H}; z_{\text{cur}}). \tag{12}$$

The active set and weights are recomputed from the current observation at each MPC solve. When no intermediate waypoint remains ahead, the auxiliary cost vanishes and planning uses only the terminal goal cost.

5 Experimental Setup

Environments: We evaluate on standard manipulation and navigation benchmarks for world models. For navigation, we use **Two-Rooms** (Sobal et al., 2025), where a red-dot agent must move from one room to a random target in an adjoining room through a single door, as well as maze navigation tasks such as **PointMaze – UMaze** and the more difficult **PointMaze – Medium** (Fu et al., 2020). For manipulation, we use **PushT** (Chi et al., 2025; Florence et al., 2022) in 2D and the single-cube OGBench-Cube task (Park et al., 2025), which we call **Cube**, in 3D. In PushT, a blue-dot agent pushes a T-shaped block to a goal pose. In Cube, a robot arm picks up and places a cube at a target. Example frames appear in Figure 5 in Appendix B. We follow the data generation evaluation setup used for each pretrained world model to which we apply our method.

Training: For each task, we trained a lightweight endpoint-conditioned Latent Residual Waypoint model on frozen world model embeddings of task trajectories. Each observation was encoded with the task's pretrained world model encoder into a 192-dimensional latent vector, for Le-WorldModel (Maes et al., 2026), or flattened $14 \times 14 \times 8$ dimensional latent spatial features, for DINO-WM (Zhou et al., 2024) and a pretrained ResNet world model (Wang et al., 2026). Training samples were drawn from contiguous 25-step segments from the world model's training set. For a segment, we sampled a current offset i and a future offset j, with $0 \leq i < j \leq 25$, and trained the model to predict z_j. We use an MLP with final layer initialized to zero so that the initial model matches linear interpolation and train using a weighted combination of Mean-Squared Error (MSE) and cosine distance using a weight of $\lambda = 0.01$.

Evaluation: For experiments with Le-WorldModel, we evaluate on ten randomly generated seeds for each environment, reporting mean and standard deviation. Unless otherwise indicated, hyperparameters for Latent Residual Waypoints were tuned on a separate set of ten validation seeds. We follow the evaluation setup in Maes et al. (2026) and use the Cross-Entropy Method (CEM) as well as gradient-based planning with Adam for planning with Le-WorldModel. For experiments with DINO-WM and ResNet world models, we tune hyperparameters on two validation seeds and evaluate on three held out evaluation seeds. Experiments with DINO-WM and ResNet follow the evaluation setup in Wang et al. (2026) using gradient-based closed-loop and open-loop MPC. In open-loop planning, the planner only plans once, at the beginning. In closed-loop planning, the planner replans after executing every 5 steps. During hyperparameter tuning, we tune the number of waypoints, weight λ_{wp}, and the power p for weighting function for future waypoints. Latent Residual Waypoints were tuned with the same search space as Linear and Linear-Snap baselines.

Additional experimental details related to architecture, training, hyperparameters, evaluation data, and latent space visualization can be found in Appendix A.

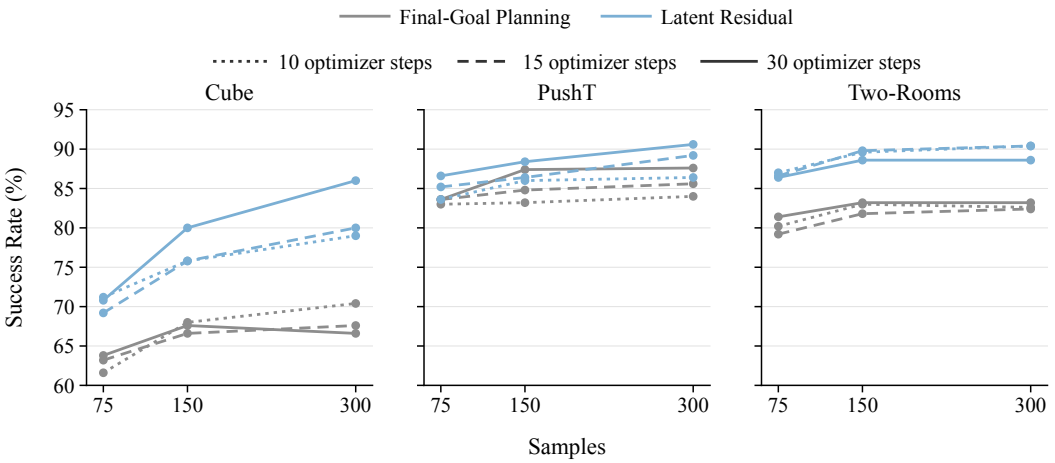


Figure 2: CEM planning performance of Latent Residual Waypoints and Final-Goal Planning as we change the number of CEM samples and optimizer steps. Additional samples allow the model to explore additional action sequences while additional optimizer steps allow the model to further refine its plan. At each fixed combination of samples and optimizer steps, Latent Residual shows higher success rate than baseline Final-Goal planning, with the largest gap on the difficult Cube task. Full results are available in Table 9 in Appendix C.1.

Baselines: To assess the quality of our approach, we compare against the following baselines:

- **Final-Goal Planning:** We compare against a planning-only baseline that uses only the terminal latent goal-matching cost, $\mathcal{C}(\hat{z}_H) = \|\hat{z}_H - z_g\|_2^2$, between the final latent rollout and the encoded goal observation $z_g = \text{enc}(o_{\text{goal}})$.
- **Linear:** We embed the start and goal observation frames into the world model latent space and produce evenly spaced latent waypoints using linear interpolation. We then use the augmented cost given by Equation 12 for planning. This tests a simpler method for producing latent waypoints between the start and goal.
- **Linear-Snap:** We first compute Linear interpolation waypoints, then snap each to its nearest dataset-frame latent embedding among up to 100,000 dataset frames sampled from each environment, encoded with the frozen world model encoder. This tests whether Linear interpolation waypoints suffer from lying off the data manifold.

6 Results

6.1 Difficult Planning Settings

Latent Residual Waypoints improve planning with constrained budgets. Intermediate latent waypoints provide useful search guidance when the planner cannot rely on large sample counts or many optimization steps. Across CEM budgets, Latent Residual Waypoints consistently improve Success Rate over Final-Goal Planning, with the largest gains on Cube and Two-Rooms (Figure 2), and more moderate gains on PushT. Under the strictest budget, 75 samples and 10 optimizer steps, Latent Residual outperforms Final-Goal Planning at the full default budget of 300 samples and 30 optimizer steps for both Cube and Two-Rooms by 4.6 percentage points and 3.8 percentage points, respectively (Table 9).

Latent Residual Waypoints improve performance on difficult planning tasks. Latent Residual Waypoints provide benefits on Cube tasks that require larger cube displacements. Since LeWorldModel Cube examples have small displacements on average (mean 0.0443), we construct harder evaluation buckets by filtering for examples above different displacement percentiles. Across all buckets, Latent Residual Waypoints outperform Final-Goal Planning with both CEM and Adam (Figure 3b). The gains range from 6.40 percentage points in the `top10` bucket to 17.00 percentage points in the `top30` bucket. Statistics for each bucket are reported in Table 10 in Appendix C.1.

Method	Cube	PushT	Two-Rooms
Final-Goal Planning	66.60 ±4.43	87.60 ±6.52	83.20 ±4.92
Linear	67.20 ±5.30	88.60 ±7.10	83.80 ±5.85
Linear-Snap	71.60 ±6.20	88.20 ±6.89	83.20 ±3.91
Latent Residual	**86.00** ± **3.30**	**90.80** ± **4.40**	**88.60** ± **5.30**

Table 1: CEM planning success rate compared with waypoint and Final-Goal baselines for Le-WorldModel.

6.2 Comparing with Simpler Waypoint Methods

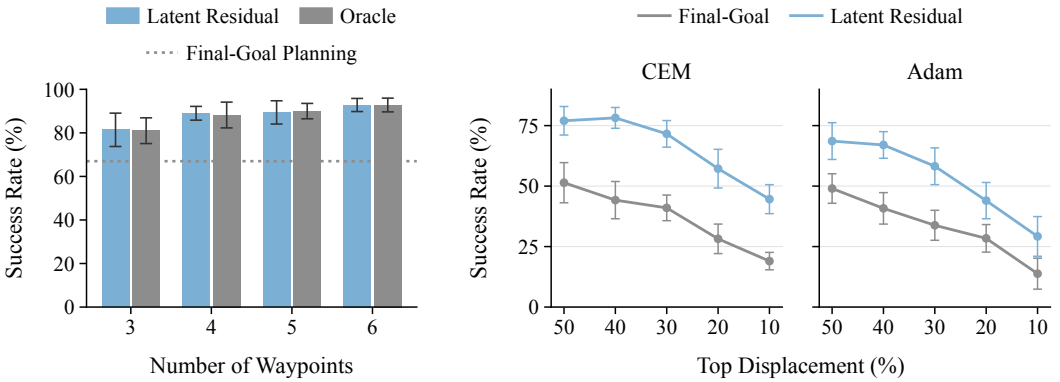
Latent Residual Waypoints improve CEM planning success. Latent Residual Waypoints outperform Final-Goal Planning and simpler waypoint baselines when used for CEM planning with Le-WorldModel across all three environments (Table 1). Tuned hyperparameters can be found in Table 4 in Appendix A. While Linear and Linear-Snap improve over Final-Goal Planning on Cube and PushT, they underperform Latent Residual Waypoints. On Cube, the hardest environment, Latent Residual Waypoints increase Success Rate from 66.60% to 86.00%, a 19.4 percentage point gain. With gradient-based planning with Adam, the story is more mixed (Table 11 in Appendix D). We again see large improvements in the difficult Cube environment, where success rate increases 11 percentage points, but on PushT and Two-Rooms Latent Residual Waypoints underperform the simpler Linear baseline. In both of those settings, though, we see little benefit from any of the waypoint methods, with 2 percentage points or fewer separating the top and bottom method.

Latent Residual Waypoints improve closed-loop MPC performance. These benefits extend to DINO-WM and ResNet world model encoders. On closed-loop MPC planning, Latent Residual Waypoints consistently perform better than baseline Final-Goal Planning (Table 2), including a 14 percentage point increase on Two-Rooms and a 6.6 percentage point increase on the more difficult maze navigation task, Medium, when planning with a pretrained ResNet world model. On the PushT task, as in the evaluations using Le-WorldModel, we tie Linear interpolation waypoints when planning with DINO-WM and underperform Linear-Snap waypoints when planning with the ResNet world model. On open-loop MPC, however, we see limited improvements and often tie or lose to Final-Goal Planning, with the exception of an 8 percentage point improvement on UMaze with DINO-WM (Table 12). Upon further analysis, we find that for both Final-Goal Planning and Latent Residual Waypoints, many failures involve overshooting the goal state and being unable to correct. This ultimately accounts for 19.90% of Final-Goal failures and 22.30% of Latent Residual failures, suggesting that the underlying issue is due to being unable to replan during open-loop planning.

Latent Residual Waypoints are competitive with oracle waypoints. On Cube, the hardest setting for Final-Goal Planning with Le-WorldModel, Latent Residual Waypoints approach the performance of waypoint planning with oracle trajectory embeddings (Figure 3a). We tune hyperparameters for each number of waypoints separately for both oracle and Latent Residual. The success rate using Latent Residual waypoints remains within 1 percentage point of that of oracle waypoints when using between 3 and 6 waypoints. At 6 waypoints, the mean success rates are exactly matched and at 4 waypoints the mean success rate for Latent Residual Waypoints is slightly higher than that of oracle waypoints.

Encoder	Planning Approach	Two-Rooms	PointMaze – UMaze	PointMaze – Medium	PushT
DINO	Final-Goal Planning	92.70 ±5.00	92.70 ±4.10	97.30 ±0.90	80.00 ±2.80
	Linear	92.70 ±5.00	92.70 ±5.20	97.30 ±0.90	**81.30** ± **2.50**
	Linear-Snap	92.70 ±5.00	95.30 ±0.90	96.00 ±0.00	78.00 ±1.60
	Latent Residual	**94.70** ± **2.50**	**96.70** ± **3.40**	**99.30** ± **0.90**	**81.30** ± **3.80**
ResNet	Final-Goal Planning	51.30 ±6.20	61.30 ±1.90	84.70 ±5.00	79.30 ±6.60
	Linear	56.70 ±4.10	62.70 ±4.10	85.30 ±2.50	80.00 ±9.90
	Linear-Snap	58.00 ±2.80	62.70 ±1.90	86.70 ±2.50	**84.00** ± **7.10**
	Latent Residual	**65.30** ± **3.80**	**63.30** ± **3.80**	**91.30** ± **1.90**	82.70 ±0.66

Table 2: Closed-Loop MPC planning success rate across models and planning approaches.



(a) Oracle waypoints comparison with Latent Residual. The dotted line shows Final-Goal Planning success rate.

(b) Evaluation on larger displacement subsets of the Cube environment, using both CEM and gradient-based Adam planning.

Figure 3: Latent Residual Waypoints evaluated on the Cube environment.

6.3 Latent Residual Waypoints as Plausible Intermediate Latents

Decoded Latent Residual Waypoints closely match the oracle dataset trajectory. Decoded Latent Residual Waypoints remain visually coherent across the trajectory and closely resemble oracle rollouts (Figure 4). In contrast, decoded Linear interpolation waypoints lose coherence mid-trajectory, suggesting that straight-line interpolation drifts away from regions of latent space corresponding to realistic observations. This qualitative difference suggests that the learned residual waypoints better follow the world-model latent manifold.

Latent Residual Waypoints are closer to real dataset embeddings. For both PushT and Cube tasks, using Le-WorldModel as the encoder, Latent Residual Waypoints reduce average L_2 distance to both the nearest embedded oracle midpoint and to nearest embedded dataset frame compared to Linear interpolation waypoints (Table 3). In the Cube task, our waypoints are 70.20% closer to the oracle midpoint and 42.20% closer to the nearest embedded dataset frame and distance reductions are even larger in PushT (77.50% and 59.30% for nearest oracle midpoint and dataset frame, respectively).

7 Conclusion

In this work, we introduce Latent Residual Waypoints, a method for generating intermediate latent subgoals for planning with pretrained latent world models. By learning residual corrections to linear interpolation, the model yields waypoints that guide evaluation-time planning. Planning with Latent Residual Waypoints is highly effective under limited planning budgets, allowing a planner to outperform full-budget planning on both Two-Rooms and Cube with only a third the number of optimizer steps and less than a third the number of CEM samples. Additionally, we show consistent improvements in CEM planning across manipulation and navigation tasks using a frozen Le-WorldModel encoder. These benefits extend to both DINO-WM and a ResNet world model, where we show planning with Latent Residual Waypoints improves success rate on Two-Rooms and maze navigation tasks. Finally, our waypoint generation produces latents that better align with plausible trajectory latents than naive interpolation, lying closer in L2 distance to encoded dataset

Task	Method	Distance to Oracle Waypoint ↓	Distance to Nearest Dataset Frame ↓
Cube	Linear	8.892	6.484
	Latent Residual	**2.650**	**3.746**
PushT	Linear	7.925	7.113
	Latent Residual	**1.783**	**2.894**

Table 3: Distance metrics for Cube and PushT.

frames. When these latents are decoded, we find that they are more coherent than decoded Linear interpolation waypoints and also seem to track the oracle trajectory frames.

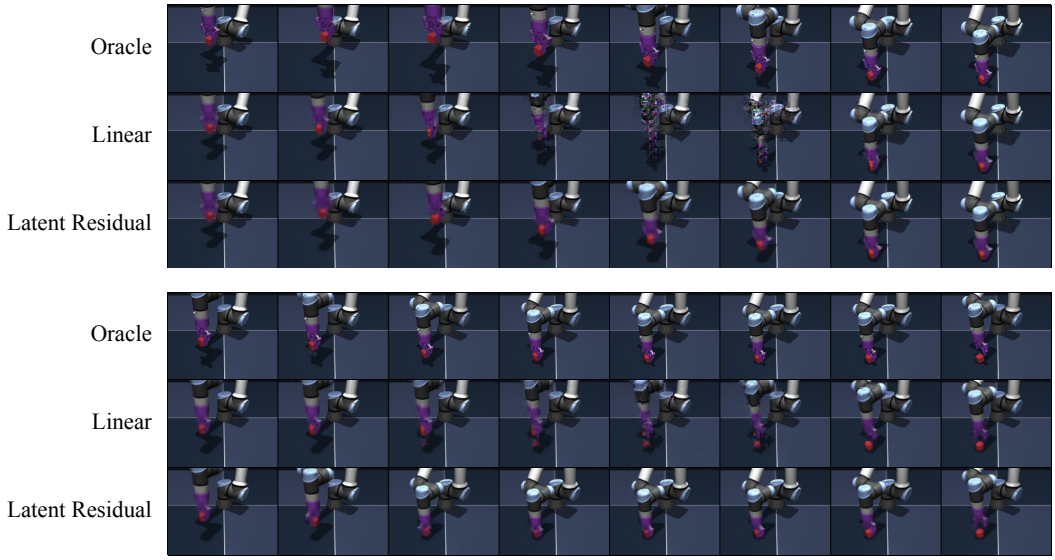


Figure 4: Visualized trajectories for the Cube environment. We compare the Oracle dataset frames, decoded Linear interpolation waypoints, and decoded Latent Residual Waypoints. Linear interpolation waypoints lose coherence in the middle of the trajectory, whereas Latent Residual Waypoints are not only coherent, they also track the Oracle dataset frames.

8 Future Work & Limitations

Our current approach has a few limitations. First, during planning we produce waypoints deterministically, which causes difficulties in tasks that admit multiple valid solutions. A promising avenue of future research may be to model multiple candidate waypoint sequences or a distribution over sequences, allowing the planner to select among different routes. Future work may also explore adaptive waypoint placement, rather than having the number and spacing of waypoints fixed in advance.

As an additional limitation, our approach introduces additional hyperparameters for tuning, such as the number of waypoints K, the weight for the minimum-over-time waypoint cost $\lambda_{\mathbf{wp}}$, as well as the power p used to weight future waypoints. Finally, although we see substantial gains in difficult settings, we see smaller gains on easier tasks. Better understanding when waypoint guidance is beneficial could lead to planners that apply waypoint costs only when they provide useful signal.

AI use statement

In this work, we used generative AI tools for coding and implementation of experiments, translating code into mathematical equations. We have not used AI tools for proving mathematical claims, generating synthetic datasets, proposing and refining hypotheses, designing or providing feedback on research methodology, or other required disclosure uses. Additionally, we used AI tools to check the writing for spelling and grammar and aid in stylistic edits. We have reviewed all AI-assisted work and examined code produced by the AI models for correctness, as well as subjected generated code to an adversarial audit by different AI tools.

Ethics statement

Our work does not involve human subjects and is unlikely to lead to harmful insights. Our work involves simple simulated environments for planning.

Reproducibility statement

We have taken several steps to ensure that our results are reproducible. Upon publication, we will release our training and evaluation code as well as the trained model checkpoints. The release will also include the configurations and hyperparameters used for the experiments reported in the paper, as well as scripts and instructions for reproducing the main experimental results.

Additionally, in the paper we include all tuned hyperparameter values as well as detailed information regarding training data and optimization settings. We also report results across multiple random seeds and show means and standard deviations.

References

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, et al. V-jepa 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

O. Brock and O. Khatib. Elastic strips: A framework for motion generation in human environments. In *IJRR*, volume 21, pp. 1031–1052, December 2002.

Joao Carvalho, An T Le, Mark Baierl, Dorothea Koert, and Jan Peters. Motion planning diffusion: Learning and planning of robot motions with diffusion models. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1916–1923. IEEE, 2023.

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. *Int. J. Rob. Res.*, 44(10–11):1684–1704, September 2025. ISSN 0278-3649.

Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

Benjamin Eysenbach, Vivek Myers, Ruslan Salakhutdinov, and Sergey Levine. Inference via interpolation: Contrastive representations provably enable planning and inference. *Advances in Neural Information Processing Systems*, 37:58901–58928, 2024.

Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on robot learning*, pp. 158–168. PMLR, 2022.

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

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

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*, pp. 2555–2565. PMLR, 2019b.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004.

Zixuan Huang, Yating Lin, Fan Yang, and Dmitry Berenson. Subgoal diffuser: Coarse-to-fine subgoal generation to guide model predictive control for robot manipulation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16489–16495, 2024. doi: 10.1109/ICRA57147.2024.10610189.

Xuhui Kang and Yen-Ling Kuo. Incorporating task progress knowledge for subgoal generation in robotic manipulation through image edits. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 7490–7499. IEEE, 2025.

Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, 2009.

L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996. doi: 10.1109/70.508439.

Yann LeCun. A Path Towards Autonomous Machine Intelligence Version. 2022.

Martina Lippi, Petra Poklukar, Michael C Welle, Anastasiia Varava, Hang Yin, Alessandro Marino, and Danica Kragic. Latent space roadmap for visual action planning of deformable and rigid object manipulation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5619–5626. IEEE, 2020.

Dongxiu Liu, Haoyi Niu, Zhihao Wang, Jinliang Zheng, Yinan Zheng, Zhonghong Ou, Jianming Hu, Jianxiong Li, and Xianyuan Zhan. Efficient robotic policy learning via latent space backward planning. In *International Conference on Machine Learning*, pp. 39147–39161. PMLR, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

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

Suraj Nair and Chelsea Finn. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1gzR2VKDH.

Fei Ni, Jianye Hao, Shiguang Wu, Longxin Kou, Jiashun Liu, Yan Zheng, Bin Wang, and Yuzheng Zhuang. Generate subgoal images before act: Unlocking the chain-of-thought reasoning in diffusion model for robot manipulation with multimodal prompts. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13991–14000, 2024. doi: 10.1109/CVPR52733.2024.01327.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. OGBench: Benchmarking offline goal-conditioned RL. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=M992mjgKzI.

Julie Porteous and Stephen Cresswell. Extending landmarks analysis to reason about resources and repetition. In *Proceedings of the 21st Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG)*, pp. 102–113, 2002.

Julie Porteous, Laura Sebastia, and Jörg Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the Sixth European Conference on Planning (ECP-01)*, pp. 37–48. Springer-Verlag, 2001.

Sean Quinlan and Oussama Khatib. Elastic bands: connecting path planning and control. *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pp. 802–807 vol.2, 1993. URL https://api.semanticscholar.org/CorpusID:5641886.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

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

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *International conference on machine learning*, pp. 4732–4741. PMLR, 2018.

Ying Wang, Oumayma Bounou, Gaoyue Zhou, Randall Balestriero, Tim GJ Rudner, Yann LeCun, and Mengye Ren. Temporal straightening for latent planning. *arXiv preprint arXiv:2603.12231*, 2026.

Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. *Advances in neural information processing systems*, 28, 2015.

Amber Xie, Oleh Rybkin, Dorsa Sadigh, and Chelsea Finn. Latent diffusion planning for imitation learning. In *International Conference on Machine Learning*, pp. 68710–68724. PMLR, 2025.

Wancong Zhang, Basile Terver, Artem Zholus, Soham Chitnis, Harsh Sutaria, Mido Assran, Randall Balestriero, Amir Bar, Adrien Bardes, Yann LeCun, et al. Hierarchical planning with latent world models. *arXiv preprint arXiv:2604.03208*, 2026.

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

Matthew Zucker, Nathan Ratliff, Anca Dragan, Mikhail Pivtoraiko, Matthew Klingensmith, Christopher Dellin, J. Andrew (drew) Bagnell, and Siddhartha Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *Proceedings of International Journal of Robotics Research*, 32 (9):1164 – 1193, August 2013.

A Additional Experimental Details

Latent Residual MLP Architecture: We use an MLP with hidden dimension 1024, depth 4, LayerNorm and GELU (Hendrycks & Gimpel, 2016) blocks, and no dropout.

Additional training details: For Le-WorldModel, we used 3,000 episodes each from the PushT and Cube tasks and 300 episodes from Two-Rooms and trained for 30,000 steps. The final checkpoint was selected using validation MSE loss. We selected the final checkpoint for PushT and Cube and the checkpoint at step 8,500 for Two-Rooms. For DINO-WM and ResNet, we selected 1,000 episodes from each task (Two-Rooms, UMaze, Medium, and PushT) and trained for 100,000 steps, again selecting the final checkpoint based on validation MSE. Models were trained using the AdamW optimizer (Loshchilov & Hutter, 2019) with learning rate $3e-4$, weight decay $1e-4$, and batch size 2048. For DINO-WM, we selected the checkpoint at step 2000 for Two-Rooms, 40,500 for PushT, 38,500 for UMaze, and 6000 for Medium. For ResNet, we selected the checkpoint at step 11,500 for Two-Rooms, 65,000 for PushT, 39,000 for UMaze, and 10,500 for Medium.

Evaluation and Validation Details: Ten percent of contiguous segments were held out for validation and checkpoints were selected by lowest validation midpoint MSE. To avoid train-test overlap, we ensure that all episodes touched by our 10 evaluation seeds were excluded before sampling training episodes.

Default Planning Hyperparameters: For Cross-Entropy Method (CEM) planning, we use 300 action samples, 30 optimization steps, topk=30, planning horizon 5, receding horizon 5, and action block 5, so the planner replans every 25 environment steps. Adam planner comparisons use 100 action samples, 30 optimization steps, and AdamW with learning rate 0.1. For closed-loop and open-loop MPC, we use AdamW with learning rate 0.1 and 100 optimization steps per replan.

Visualization: For qualitative latent-state visualizations, we train a task-specific auxiliary decoder from frozen Le-WorldModel projected CLS latents to RGB observations. We sample frames from the task dataset, encode each frame with the task's frozen, pretrained Le-WorldModel object checkpoint, and train the decoder to reconstruct the corresponding image. The input latent is the projected CLS token with dimension 192. Encoder inputs use ImageNet normalization, while reconstruction targets used RGB values scaled to $[0, 1]$. We optimized pixel reconstruction loss with AdamW for 100 epochs using learning rate $3e-4$, weight decay $1e-4$, and batch size 192. We train using a weighted combination of MSE and L_1 loss.

Planning hyperparameters for CEM: Table 4 shows the tuned hyperparameters for the Le-WorldModel CEM experiments.

Task	Method	K	λ	Weighting	p
Cube	Linear	8	0.10	Inverse distance	1.0
	Linear-Snap	8	0.70	Uniform	—
	Latent Residual	8	1.00	Inverse distance	1.0
	Oracle	8	2.00	Inverse distance	0.5
PushT	Linear	4	0.03	Inverse distance	2.0
	Linear-Snap	4	0.03	Inverse distance	0.5
	Latent Residual	5	0.50	Inverse distance	2.0
Two-Rooms	Linear	4	0.03	Inverse distance	0.5
	Linear-Snap	4	0.70	Uniform	—
	Latent Residual	5	1.00	Uniform	—

Table 4: Hyperparameters used for CEM planning with Le-WorldModel.

Planning hyperparameters for Adam: Table 5 shows the tuned hyperparameters for the Adam planning experiments with Le-WorldModel.

Task	Method	K	λ	Weighting	p
Cube	Linear	8	0.03	Inverse distance	2.0
	Linear-Snap	4	0.50	Inverse distance	1.0
	Latent Residual	8	1.00	Inverse distance	0.5
PushT	Linear	8	0.10	Inverse distance	2.0
	Linear-Snap	8	0.30	Inverse distance	1.0
	Latent Residual	8	0.50	Uniform	—
Two-Rooms	Linear	8	0.03	Uniform	—
	Linear-Snap	3	0.70	Uniform	—
	Latent Residual	8	0.1	Inverse distance	0.5

Table 5: Hyperparameters for Adam planning with Le-WorldModel.

Planning hyperparameters for closed-loop MPC: In Table 6 we show tuned hyperparameters for closed-loop MPC DINO-WM experiments and in Table 7 we show hyperparameters for ResNet.

Task	Method	K	λ	Weighting	p
Medium	Linear	8	4.00	Uniform	—
	Linear-Snap	12	2.00	Uniform	—
	Latent Residual	8	1.00	Uniform	—
PushT	Linear	8	0.10	Uniform	—
	Linear-Snap	8	0.10	Inverse distance	0.5
	Latent Residual	8	0.10	Inverse distance	0.5
UMaze	Linear	8	4.00	Uniform	—
	Linear-Snap	8	4.00	Uniform	—
	Latent Residual	16	2.00	Uniform	—
Two-Rooms	Linear	3	0.10	Inverse distance	1.0
	Linear-Snap	8	0.10	Inverse distance	0.5
	Latent Residual	3	1.00	Inverse distance	1.0

Table 6: Hyperparameters for closed-loop MPC with DINO-WM.

Task	Method	K	λ	Weighting	p
Medium	Linear	5	0.50	Uniform	—
	Linear-Snap	8	0.50	Uniform	—
	Latent Residual	5	0.50	Uniform	—
PushT	Linear	5	0.50	Uniform	—
	Linear-Snap	3	0.25	Inverse distance	1.0
	Latent Residual	8	0.05	Inverse distance	1.0
UMaze	Linear	8	0.10	Inverse distance	0.5
	Linear-Snap	12	1.00	Uniform	—
	Latent Residual	3	0.50	Uniform	—
Two-Rooms	Linear	8	4.00	Uniform	—
	Linear-Snap	16	2.00	Uniform	—
	Latent Residual	12	4.00	Uniform	—

Table 7: Hyperparameters for closed-loop MPC experiments with ResNet.

Planning hyperparameters for open-loop MPC: In Table 8 we show the tuned hyperparameters for open-loop MPC experiments.

Task	Encoder	K	λ	Weighting	p
Medium	DINO	8	1.00	Uniform	—
	ResNet	5	0.50	Uniform	—
PushT	DINO	8	0.10	Inverse distance	0.5
	ResNet	3	0.10	Inverse distance	1.0
UMaze	DINO	16	2.00	Uniform	—
	ResNet	3	0.50	Uniform	—
Two-Rooms	DINO	3	1.00	Inverse distance	1.0
	ResNet	12	4.00	Uniform	—

Table 8: Hyperparameters for Latent Residual Waypoints on open-loop MPC for both DINO-WM and ResNet.

B Details on Environments

In Figure 5 we visualize the tasks we use for training and evaluation.

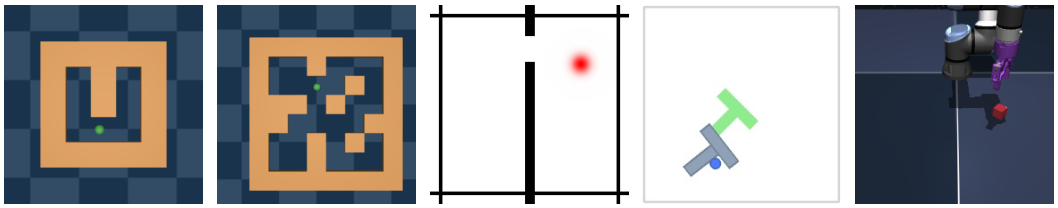


Figure 5: An example frame from each task we use for training and evaluation.

C Additional Results

C.1 Difficult Evaluation Settings

We show additional data related to our experiments in difficult planning settings. Table 9 shows the underlying data for Figure 2. Latent Residual outperforms the Final-Goal Planning baseline in almost all instances. Latent Residual is most helpful in the Cube task, followed by Two-Rooms. It provides only slight benefits in PushT. This table reports an independent run from the main CEM table, and overlapping configurations may differ slightly due to random variation. Table 10 provides additional information on our splits for the displacement-filtered Cube task. We show the number of candidate examples retained within each bucket and the minimum displacement for tasks in that bucket.

D Adam Planning with Le-WorldModel

In Table 11 we show results for Adam planning with Le-WorldModel. Hyperparameters can be found in Appendix A. On Cube, Latent Residual Waypoints outperform all baselines, but they lose to the Linear baseline on PushT and Two-Rooms.

D.1 Open Loop MPC

In Table 12 we show results for open-loop MPC planning. Hyperparameters can be found in Appendix A.

Dataset	CEM Samples	CEM Optimizer Steps	Final-Goal Planning	Latent Residual
Cube	75	10	61.60 ±6.00	**71.20 ± 4.00**
		15	63.20 ±4.10	**69.20 ± 5.00**
		30	63.80 ±4.30	**70.80 ± 4.60**
	150	10	68.00 ±4.30	**75.80 ± 3.90**
		15	66.60 ±5.60	**75.80 ± 4.70**
		30	67.60 ±6.10	**80.00 ± 4.30**
	300	10	70.40 ±6.60	**79.00 ± 4.30**
		15	67.60 ±4.00	**80.00 ± 5.20**
		30	66.60 ±4.40	**86.00 ± 3.30**
PushT	75	10	83.00 ±7.30	**83.60 ± 5.30**
		15	83.60 ±6.90	**85.20 ± 6.50**
		30	83.60 ±5.10	**86.60 ± 3.70**
	150	10	83.20 ±4.90	**86.00 ± 6.90**
		15	84.80 ±6.60	**86.40 ± 4.70**
		30	87.40 ±5.10	**88.40 ± 4.50**
	300	10	84.00 ±6.30	**86.40 ± 5.50**
		15	85.60 ±5.30	**89.20 ± 6.50**
		30	87.60 ±6.90	**90.60 ± 4.00**
Two-Rooms	75	10	80.20 ±5.00	**87.00 ± 4.50**
		15	79.20 ±5.90	**86.60 ± 3.40**
		30	81.40 ±6.30	**86.40 ± 4.60**
	150	10	83.00 ±4.50	**89.60 ± 4.40**
		15	81.80 ±4.60	**89.80 ± 4.30**
		30	83.20 ±6.20	**88.60 ± 3.00**
	300	10	82.60 ±4.30	**90.40 ± 4.60**
		15	82.40 ±5.30	**90.40 ± 4.00**
		30	83.20 ±4.90	**88.60 ± 5.30**

Table 9: Full comparison of Final-Goal Planning and Latent Residual Waypoints Success Rate, reported as Success Rate percentages.

Bucket	Retained Fraction	Candidate Starts	Min. Displacement
`top50`	50%	880,000	0.0113
`top40`	40%	704,000	0.0178
`top30`	30%	528,000	0.0332
`top20`	20%	352,000	0.0645
`top10`	10%	176,000	0.1416

Table 10: Candidate starting points and minimum displacement for each filtered bucket.

Method	Cube	PushT	Two-Rooms
Final-Goal Planning	67.00 $\pm$4.80	83.20 $\pm$4.30	93.20 $\pm$1.70
Linear	69.00 $\pm$4.50	**85.00 $\pm$ 5.60**	**95.20 $\pm$ 3.00**
Linear-Snap	69.00 $\pm$5.30	82.40 $\pm$4.30	92.40 $\pm$2.60
Latent Residual	**78.00 $\pm$ 6.40**	84.60 $\pm$4.80	93.60 $\pm$3.00

Table 11: Adam planning success rate compared with waypoint baselines.

Encoder	Planning Approach	Two-Rooms	PointMaze – UMaze	PointMaze – Medium	PushT
DINO	Final-Goal Planning	$\mathbf{84.70 \pm 9.00}$	57.30 ± 6.60	$\mathbf{79.30 \pm 3.40}$	$\mathbf{66.70 \pm 4.70}$
	Latent Residual	84.00 ± 6.50	$\mathbf{65.30 \pm 1.90}$	76.70 ± 2.50	$\mathbf{66.70 \pm 4.70}$
ResNet	Final-Goal Planning	$\mathbf{24.70 \pm 4.10}$	12.00 ± 5.90	60.70 ± 6.80	$\mathbf{68.70 \pm 6.20}$
	Latent Residual	24.00 ± 3.30	$\mathbf{14.00 \pm 8.20}$	$\mathbf{62.70 \pm 4.10}$	$\mathbf{68.70 \pm 6.20}$

Table 12: Open-loop MPC results for Latent Residual Waypoints and Final-Goal Planning.