BEYOND ONE-STEP ACCURACY: STATE-AFFINE LATENT TRANSITION FOR RELIABLE VISUAL PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Joint-embedding world models enable visual planning by learning action-conditioned dynamics in latent space. Yet they are commonly trained for one-step prediction on encoded states, while planning recursively applies the learned transition to its own predictions. One-step accuracy therefore does not capture how prediction errors propagate under recursive rollout. We decompose multi-step rollout error into the errors introduced at individual steps and their propagation through subsequent transitions. We show that state-affine dynamics are precisely the differentiable transitions with state-independent Jacobians, eliminating the nonlinear propagation residual and making the error propagation operators depend only on the action sequence. Guided by this result, we introduce SALT (**S**tate-**A**ffine **L**atent **T**ransition), an action-conditioned state-affine dynamics model in which the action modulates both the state transformation and the additive update. We train SALT through recursive multi-step rollout supervision, feeding each predicted latent state back into the transition so that training matches how the model is used during planning. Across four visual planning environments, SALT exhibits 1.48–$2.19\times$ higher one-step prediction error than the matched LeWM baseline, yet improves closed-loop success in every environment by 10.0 percentage points on average. On OGBench-Cube, the fraction of episodes that fail with a sharp rise in model-predicted cost after execution decreases from 23.3% to 2.0%.

1 INTRODUCTION

Planning from visual observations requires a model that predicts how candidate actions will change the state of the environment. Joint-embedding world models learn these dynamics in a low-dimensional latent space without reconstructing pixels (Assran et al., 2023; Balestriero & LeCun, 2025; Maes et al., 2026). In latent planning, a planner evaluates candidate action sequences by applying the learned transition model recursively (Hafner et al., 2019; Sobal et al., 2026; Zhou et al., 2025). Training, however, often relies on a one-step prediction objective defined on encoded states, as in LeWM (Maes et al., 2026). Such supervision does not constrain how prediction errors propagate when the model recursively uses its own predictions. A more accurate one-step predictor can therefore be a less effective planning model.

During recursive rollout, each transition introduces a new prediction error, while earlier errors are transformed by later transitions (Asadi et al., 2019; Somalwar et al., 2025). One-step prediction error measures the deviation introduced by a transition from an encoded state, but does not describe how subsequent transitions transform that deviation. Over multiple steps, rollout error therefore depends on both the errors introduced along the trajectory and their propagation through the learned dynamics. The challenge is therefore to learn latent dynamics that remain effective for planning as prediction errors are fed back through the model.

To study this effect, we decompose multi-step rollout error into errors introduced at individual steps and their subsequent propagation. For nonlinear state transitions, this propagation involves state-dependent Jacobians and residual terms beyond the first-order approximation. We therefore ask which transition structure makes the propagation operator independent of the rollout state. We show that, among differentiable transitions, this property holds precisely when the predictor is affine in the latent state. In this case, the nonlinear residual vanishes, and the propagation operators are expressed exactly through products of action-conditioned transition matrices. For recursive planning, this pro-

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.

vides a way to separate the accuracy of individual predictions from how their errors are transformed along a candidate action sequence, motivating a structural approach to transition design.

Guided by this analysis, we introduce SALT, an action-conditioned transition model that remains affine in the latent state. The action modulates both the state transformation and the additive update, allowing different actions to induce different dynamics while keeping the state Jacobian independent of the latent state. This structural choice determines the form of error propagation, but does not by itself align one-step supervision with recursive prediction. We therefore train SALT with a multi-step rollout objective, feeding each predicted latent state back into the transition and supervising the resulting trajectory. The state-affine structure determines the form of error propagation, while rollout training aligns supervision with the recursive feedback used in planning.

Across four visual planning environments, SALT has 1.48–$2.19\times$ the one-step prediction error of the matched LeWM reproduction, yet achieves a higher mean closed-loop success rate on each environment. Its average success rate reaches 92.7%, exceeding the baseline by 10.0 percentage points. These results demonstrate that lower one-step prediction error alone does not determine better planning performance. On OGBench-Cube, the rate of failures characterized by sharply higher model-predicted costs after execution and re-observation decreases from 23.3% of episodes for LeWM to 2.0% for SALT. In these cases, plans look promising before execution but no longer do once the model sees the result.

2 Related Work

Latent World Models for Visual Planning. Latent world models support planning and control by learning compact predictive representations of visual observations. Early work learned latent state-space dynamics for prediction and control from images (Watter et al., 2015; Karl et al., 2017). World Models, PlaNet, and subsequent latent-imagination approaches further established learned latent dynamics as an effective basis for control from pixels (Ha & Schmidhuber, 2018; Hafner et al., 2019; 2021; 2025). More recent methods target visual planning directly. PLDM learns latent dynamics from reward-free offline data (Sobal et al., 2026), DINO-WM plans over pretrained visual features (Zhou et al., 2025), LeWorldModel (LeWM) jointly learns visual representations and action-conditioned latent dynamics (Maes et al., 2026), and TD-MPC2 optimizes trajectories in learned latent dynamics (Hansen et al., 2024). Related work examines reliability and efficiency in closed-loop use (Duan et al., 2024; Zhang et al., 2026; Chun et al., 2026), while large-scale video JEPA models learn predictive representations for physical understanding and planning (Assran et al., 2025). These advances demonstrate the value of latent prediction for planning, but planning performance alone does not characterize how errors propagate through repeated applications of the learned transition. We characterize how repeated composition of the learned transition transforms prediction errors, separating the errors introduced at each step from their transformation by subsequent transitions.

Stable and Structured Joint-Embedding World Models. Joint-embedding predictive learning aims to learn stable and structured representations without pixel-level reconstruction. I-JEPA learns image representations by predicting target-block embeddings from a context block (Assran et al., 2023). LeJEPA introduced SIGReg to stabilize joint-embedding learning through distributional regularization (Balestriero & LeCun, 2025), and LeWM applies it to end-to-end visual world modeling (Maes et al., 2026). Recent extensions mainly modify representation learning through quantile matching, subspace regularization, latent decomposition, or inverse-dynamics supervision (Yu et al., 2026; Zhao et al., 2026; Thil et al., 2026; Ivashkov et al., 2026), with related objectives also studied outside the joint-embedding setting (Sun et al., 2024; Wang et al., 2025b). These representation objectives do not by themselves enforce a state-independent transition Jacobian. Beyond representation learning, Terver et al. (2026) study training and planning design choices for joint-embedding world models and analyze autoregressive error growth using Lipschitz bounds. We focus on the structural condition under which the state Jacobian is independent of the latent state, yielding an exact error decomposition for action-conditioned state-affine transitions.

Multi-Step Prediction and Structured Latent Dynamics. Multi-step learning extends prediction supervision along latent trajectories (Karl et al., 2017; Hafner et al., 2019). Work on compounding model error further studies how multi-step objectives, temporal abstraction, and uncertainty-aware rollouts affect prediction over longer horizons (Asadi et al., 2019; Somalwar et al., 2025; Gumb-

sch et al., 2024; Wang et al., 2024; 2025a). However, supervising longer rollouts does not by itself enforce a state-independent transition Jacobian. Structured dynamics constrain the transition directly. Locally linear models allow the transition matrices to vary with the state (Watter et al., 2015), while softly state-invariant models regularize the dependence of latent state changes on the current state (Saanum et al., 2024). Koopman methods seek representations with linear or bilinear dynamics for prediction and control (Brunton et al., 2016; Takeishi et al., 2017; Lusch et al., 2018; Shi & Meng, 2022; Huang et al., 2023; Zhao et al., 2024; Chen et al., 2024). For fixed actions, linear and bilinear transitions have state-independent Jacobians and belong to the state-affine family. We derive this family from the Jacobian requirement and express its rollout error exactly through action-conditioned propagation matrices. We instantiate this family in a visual joint-embedding world model, learning the encoder and transition jointly from pixels through recursive rollout supervision.

3 METHOD

3.1 PROBLEM SETUP

A joint-embedding predictive architecture models future dynamics in a low-dimensional latent space rather than directly in pixel space. Let o_t denote the image observation at time t and $a_t \in \mathbb{R}^{d_a}$ the control input. The encoder ϕ, including its projection head, maps an observation to the latent state $z_t = \phi(o_t) \in \mathbb{R}^d$. The action encoder ψ maps the control input to an action condition $c_t = \psi(a_t) \in \mathbb{R}^d$. The predictor F predicts the next latent state from (z_t, c_t). We denote by $\hat{z}_t$ a latent state generated during autoregressive rollout.

A common training objective for latent transition models is one-step prediction:

$$\mathcal{L}_{\text{pred}} = \mathbb{E}_t \big\| F(z_t, c_t) - z_{t+1} \big\|^2. \tag{1}$$

The predictor is evaluated at latent states encoded from observed trajectories. Equation 1 therefore directly supervises transitions from encoded states rather than states generated by previous predictions.

At planning time, the transition model is applied recursively. Given the current latent state z_0 and a goal observation o_g with embedding $z_g = \phi(o_g)$, the planner optimizes an action sequence over a horizon of H steps:

$$\begin{aligned} a^*_{0:H-1} = \arg\min_{a_{0:H-1}} \ & \big\| \hat{z}_H - z_g \big\|^2 \\ \text{subject to} \quad \hat{z}_0 = z_0, \qquad & \hat{z}_k = F(\hat{z}_{k-1}, c_{k-1}), \qquad c_{k-1} = \psi(a_{k-1}), \quad k = 1, \dots, H. \end{aligned} \tag{2}$$

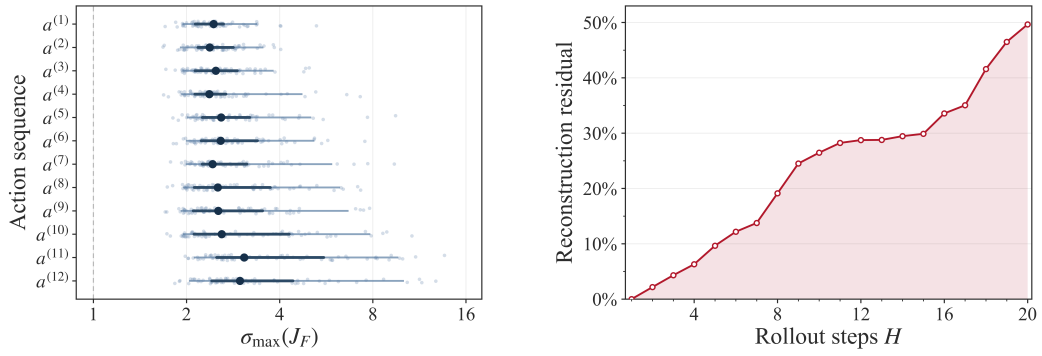
We optimize equation 2 using the cross-entropy method (CEM). Each iteration samples S candidate action sequences, evaluates their latent rollouts, and updates the sampling distribution using the sequences with the lowest terminal costs. No new observation is available during a model rollout to correct the predicted states. After executing the selected sequence, the planner replans from the new observation.

Equation 1 evaluates individual transitions from encoded states, whereas planning depends on the composition $F(\cdot, c_{H-1}) \circ \cdots \circ F(\cdot, c_0)$ along a predicted trajectory. The resulting rollout error depends on both the errors introduced at individual steps and their propagation through subsequent transitions.

3.2 RECURSIVE ERROR PROPAGATION

We distinguish two errors during recursive rollout. The one-step error $\varepsilon_t = F(z_{t-1}, c_{t-1}) - z_t$ is the deviation incurred by a single transition from the encoded state z_{t-1}. Its squared norm is the per-transition loss in equation 1. The rollout error $\delta_t = \hat{z}_t - z_t$ measures the accumulated deviation from the encoded reference trajectory after t recursive prediction steps, starting from $\hat{z}_0 = z_0$. Substituting $\hat{z}_{t-1} = z_{t-1} + \delta_{t-1}$ into the definition of δ_t and expanding the predictor around the encoded state z_{t-1} gives

$$\begin{aligned} \delta_t &= F(\hat{z}_{t-1}, c_{t-1}) - z_t \\ &= F(z_{t-1} + \delta_{t-1}, c_{t-1}) - F(z_{t-1}, c_{t-1}) + F(z_{t-1}, c_{t-1}) - z_t \\ &= J_t \, \delta_{t-1} + r_t + \varepsilon_t, \end{aligned} \tag{3}$$



(a) State dependence of the Jacobian.

(b) Reconstruction discrepancy versus horizon.

Figure 1: **State dependence and current-state rollout reconstruction of the LeWM predictor.** (a) Distribution of $\sigma_{\max}(\partial_z F)$ for the current-state Jacobian across 12 fixed action sequences while varying the current latent state. Points denote samples, the thick bar shows the interquartile range, and the dashed line marks $\sigma_{\max} = 1$. (b) Relative discrepancy between the rollout error and its current-state first-order reconstruction as the rollout horizon increases. The reconstruction uses only current-state Jacobians and omits propagation through earlier history inputs.

where $J_t = \partial_z F(z_{t-1}, c_{t-1})$ is the state Jacobian evaluated at z_{t-1}, and r_t is the residual beyond the first-order linearization.

Proposition 1 (Multi-step error decomposition)**.** *Let F be any predictor that is differentiable in the state. Then, for a rollout of H steps from $\hat{z}_0 = z_0$, the rollout error can be written as*

$$\delta_H = \sum_{j=1}^{H} \Phi_{j\to H}\big(\varepsilon_j + r_j\big), \qquad \Phi_{j\to H} = J_H J_{H-1} \cdots J_{j+1}, \qquad \Phi_{H\to H} = I. \tag{4}$$

If, in addition, $\partial_z F(\cdot, c)$ is L-Lipschitz in the state with the same constant L for every action condition c, then the residual satisfies $\|r_t\| \le \frac{L}{2}\|\delta_{t-1}\|^2$.

The proof is given in Appendix A.1. The operator $\Phi_{j\to H}$ describes how an error introduced at step j is subsequently transformed by all later transitions before reaching step H.

Equation 4 separates two factors in multi-step prediction: the error introduced by individual transitions and the propagation of those errors through later transitions.

For nonlinear transitions, r_j need not vanish, and the Jacobians in $\Phi_{j\to H}$ can vary with the encoded reference states at which they are evaluated. The propagation operator can therefore depend on the state trajectory as well as the action sequence.

We examine this state dependence in LeWM (Maes et al., 2026) by fixing the action sequence and varying the current latent state. Since LeWM conditions on a short latent history, we differentiate with respect to its most recent latent input while holding the earlier history inputs fixed when computing each Jacobian. Figure 1a shows that the largest singular value of this Jacobian varies substantially with the current latent state.

For this diagnostic, we form $\Phi_{j\to H}$ from products of these current-state Jacobians and measure the relative reconstruction discrepancy

$$\rho_H = \frac{\left\| \delta_H - \sum_{j=1}^{H} \Phi_{j\to H}\varepsilon_j \right\|}{\|\delta_H\|}. \tag{5}$$

As shown in Figure 1b, this discrepancy increases from about 10% at $H = 5$ to about 49% at $H = 20$. The reconstruction excludes propagation through earlier history inputs, so the discrepancy can contain both nonlinear effects in the current-state channel and contributions from the omitted history channels. Thus, the current-state linear reconstruction leaves an increasing fraction of the rollout error unexplained over the tested horizons.

3.3 STATE-AFFINE DYNAMICS

We seek a transition family whose propagation operators are determined by the action sequence at every rollout step. We therefore require the state Jacobian to be independent of the latent state. For differentiable transitions, this requirement is equivalent to an affine dependence on the latent state.

Proposition 2 (Characterization of state-independent Jacobians)**.** *For each fixed action condition c, let $F(\cdot, c) : \mathbb{R}^d \to \mathbb{R}^d$ be differentiable. Its state Jacobian is independent of z if and only if $F(\cdot, c)$ is affine:*

$$F(z, c) = \mathcal{A}(c)\, z + \mathcal{B}(c). \tag{6}$$

The proof is given in Appendix A.2.

Under equation 6, any state deviation δ is propagated exactly as

$$F(z + \delta, c) - F(z, c) = \mathcal{A}(c)\delta.$$

Consequently, equation 3 gives $r_t \equiv 0$ and $J_t = \mathcal{A}(c_{t-1})$. The multi-step error decomposition therefore reduces to

$$\delta_H = \sum_{j=1}^{H} \Phi_{j\to H}\, \varepsilon_j, \qquad \Phi_{j\to H} = \mathcal{A}(c_{H-1})\mathcal{A}(c_{H-2}) \cdots \mathcal{A}(c_j). \tag{7}$$

For a fixed action sequence, the propagation matrices in equation 7 can be computed directly from $\mathcal{A}(c_k)$ without generating the latent rollout. We use this property for analysis and diagnostics, while planning uses the CEM procedure defined in Section 3.1.

3.4 ACTION-CONDITIONED STATE-AFFINE TRANSITION

Proposition 2 requires an affine dependence on the latent state but does not prescribe the functional forms of $\mathcal{A}(c)$ and $\mathcal{B}(c)$. We use the following parameterization:

$$F(z, c) = \mathcal{A}(c)\, z + Bc + b, \qquad \mathcal{A}(c) = \mathcal{A}_0 + \sum_{r=1}^{R} \left(W_g c\right)_r N_r, \tag{8}$$

where $\mathcal{A}_0, N_r, B \in \mathbb{R}^{d\times d}$, $W_g \in \mathbb{R}^{R\times d}$, and $b \in \mathbb{R}^d$. The coefficient $(W_g c)_r$ weights the modulation matrix N_r, while $Bc + b$ parameterizes $\mathcal{B}(c)$. This allows actions to change the state transformation itself while preserving affine dependence on the latent state. Section F.1 evaluates the role of this action-dependent modulation.

We constrain $\sigma_{\max}(\mathcal{A}_0) < 1$ to limit the gain of the shared base transition. Section B.3 provides the parameterization and initialization details.

3.5 RECURSIVE ROLLOUT TRAINING

The state-affine structure makes recursive error propagation explicit, but it does not by itself resolve the mismatch between one-step supervision and the recursive use of the transition at planning time. We therefore train SALT with a recursive multi-step rollout objective.

Let K denote the training rollout length, which is distinct from the planning horizon H. Starting from $\hat{z}_0 = z_0$, the transition is applied recursively as

$$\hat{z}_k = F(\hat{z}_{k-1}, c_{k-1}),$$

and every predicted state is supervised:

$$\mathcal{L}_{\text{roll}} = \mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} \left\|\hat{z}_k - z_k\right\|^2\right]. \tag{9}$$

During training, predicted latents are fed back into the transition, matching the recursive use of the model during planning. We use $K = 5$, supervising all five predictions from a six-frame training window.

Table 1: **Closed-loop planning success rates (%) across four environments.** Controlled evaluations use $H = 5$ and report the mean±standard deviation over three evaluation seeds of 50 episodes each. † quoted from the literature under different protocols. Bold marks the best mean among controlled rows.

Method	Two-Room	Reacher	PushT	OGBench-Cube	Avg.
PLDM†	97	78	78	65	79.50
DINO-WM†	100	79	74	86	84.75
Sub-JEPA†	95.00	84.00	89.00	76.33	86.08
SD-JEPA†	90.00	88.00	97.30	72.00	86.83
SMWM†	99.00	66.00	83.00	84.00	83.00
Random	2.00±2.00	12.00±4.00	5.30±2.30	42.00±4.00	15.33
LeWM	90.67±4.60	85.33±2.30	87.33±4.71	67.33±4.20	82.67
SALT	**98.00**±2.00	**90.70**±3.10	**88.00**±5.20	**94.00**±2.00	**92.68**

The training objective combines recursive rollout supervision with SIGReg:

$$\mathcal{L} = \mathcal{L}_{\text{roll}} + \lambda \mathcal{R}_{\text{SIGReg}}, \tag{10}$$

Here, $\mathcal{R}_{\text{SIGReg}}$ regularizes the encoded latents from all frames in the training window, with $\lambda = 0.09$. SIGReg discourages representation collapse by encouraging the latent distribution to match an isotropic Gaussian (Balestriero & LeCun, 2025). We jointly optimize the encoder, action encoder, and transition, retaining gradients through the target latents as in LeWM (Maes et al., 2026).

The state-affine structure determines the form of error propagation, while rollout supervision trains the model under the recursive feedback used in planning.

4 Experiments

4.1 Experimental Setup

We evaluate SALT on four visual planning environments, following the benchmark setting of LeWM (Maes et al., 2026): Two-Room (Sobal et al., 2025), Reacher (Tassa et al., 2018), PushT (Chi et al., 2023; Zhou et al., 2025), and OGBench-Cube (Park et al., 2025). Planning is performed with CEM (Rubinstein & Kroese, 2004), using the MSE between the predicted terminal latent state and the goal embedding as the terminal cost. The planning horizon and the replanning interval are both set to 5, so the planner replans from a new observation only after the entire planned sequence has been executed. For each environment, we evaluate one trained model per method on three evaluation seeds of 50 episodes each, and report the mean and standard deviation across seeds. Each goal is the observation 25 environment steps after the initial state in the same offline trajectory. All experiments conducted in our implementation use a single NVIDIA H100 80 GB GPU. Additional environment, training, and evaluation details are provided in Appendix B.

For controlled comparison with the reproduced baseline, the encoder architecture, action encoder, representation regularizer, and planner are matched between the two models. The models differ in their transition structure and prediction training configuration, whose combinations are evaluated in Table 3.

We evaluate Random, reproduced LeWM, and SALT under a shared protocol. Reproduced LeWM serves as the controlled baseline for all paired comparisons. Results for PLDM (Sobal et al., 2026), DINO-WM (Zhou et al., 2025), Sub-JEPA (Zhao et al., 2026), SD-JEPA (Thil et al., 2026), and SMWM (Ivashkov et al., 2026) are taken from their respective papers and reported separately in Table 1 to provide context.

4.2 Closed-Loop Planning Performance

Table 1 reports closed-loop planning success rates on the four environments.

Table 2: **Predictor complexity and computational efficiency.** Parameter counts and single-step forward times cover the predictor together with its projection head. Planning time is the CEM wall-clock time averaged over the four environments and $H \in \{5, 10, 15, 20\}$. Additional timing details are provided in Appendix B.6.

Predictor	Params	Forward (ms)	Training (it/s)		Planning (s)
			One-step	Rollout	
LeWM (ViT-S)	11.58M	2.36	6.5	2.9	35.2
SALT (Affine)	0.70M (×16.5)	0.29 (×8.1)	6.0	2.4	11.4 (×3.1)

Planning Performance Across Four Environments. SALT reaches an average success rate of 92.7%, exceeding reproduced LeWM by 10.0 percentage points under the matched evaluation protocol. Its mean success rate is higher in all four environments. The largest gain occurs on OGBench-Cube, where success increases from 67.3% to 94.0%, a gain of 26.7 percentage points.

Inference and Planning Efficiency. Table 2 compares model size and computational efficiency. Counting the predictor and its projection head where present, SALT uses 0.70 M parameters, compared with 11.58 M for LeWM. Relative to LeWM, SALT achieves an $8.1\times$ speedup in single-step prediction and a $3.1\times$ speedup in CEM planning. This compact transition reduces the cost of repeatedly evaluating candidate action sequences, a central computation in model-based planning.

4.3 Why One-Step Accuracy Fails to Predict Planning Quality

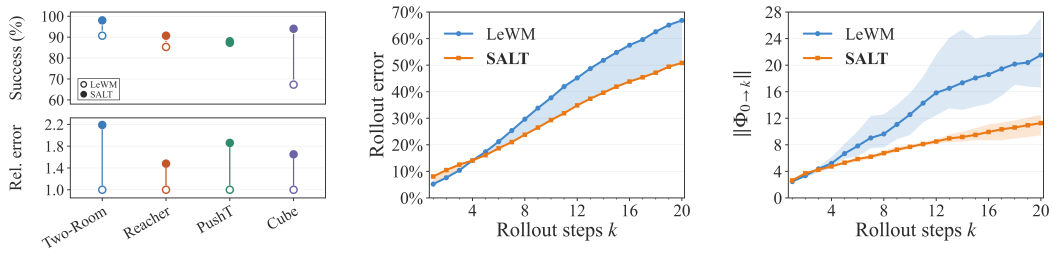
SALT achieves better closed-loop planning performance than LeWM despite having a higher one-step prediction error. We now ask what explains this apparent contradiction. Figure 2 examines three increasingly deployment-relevant quantities: one-step prediction error, recursive rollout error, and the propagation of errors through subsequent transitions.

One-Step Error Is Not a Reliable Proxy for Planning Quality. As shown in Figure 2a, the one-step prediction error of SALT is $1.48\times$–$2.19\times$ that of LeWM across the four environments, yet SALT achieves a higher mean closed-loop success rate in every case. Moreover, the relative one-step error does not track the planning gain: Two-Room has the largest relative prediction error but only an intermediate planning gain, whereas OGBench-Cube has one of the smallest relative errors and the largest gain. Thus, in the evaluated setting, one-step prediction error alone is not a reliable proxy for closed-loop planning quality.

Rollout Error Alone Does Not Explain the Planning Gap. As shown in Figure 2b, the rollout-error ranking of the two models reverses during recursive prediction. On PushT, SALT has higher rollout error than LeWM over the first four steps and lower error thereafter. Over twenty steps, rollout error increases by factors of 6.29 and 12.83 for SALT and LeWM, respectively. The four-environment measurements in Section C.6 further show that the models have similar rollout errors on OGBench-Cube over much of the evaluated horizon, despite the 26.7 percentage point planning gap. These observations indicate that rollout-error magnitude alone does not fully characterize the difference in planning performance.

Evidence for Milder Recursive Error Propagation. As shown in Figure 2c, the measured propagation-operator norms on PushT are comparable at the first step but grow at different rates thereafter. Over twenty steps, the norm increases by a factor of 4.28 for SALT and 8.74 for LeWM, and the ranges over nine measurements no longer overlap after the fourth step. Consistent with the exact state-affine decomposition in equation 7, the relative reconstruction discrepancy of SALT remains at the 10^{-6} level across rollout steps. In contrast, the current-state first-order reconstruction of LeWM becomes increasingly incomplete with horizon, as detailed in Appendix C. These diagnostics are consistent with milder propagation in the measured state channel, alongside the planning advantage of SALT.

Interaction Between Transition Structure and Rollout Training. Table 3 reports mean success across the four environments for the four combinations of transition structure and training objective,



(a) One-step error vs. success. (b) Rollout error growth. (c) Propagation-operator norm.

Figure 2: **From one-step accuracy to recursive error propagation.** (a) Relative one-step prediction error and closed-loop success across four environments. (b) Recursive rollout error on PushT. (c) Spectral norm of the measured propagation operator on PushT; the band spans the minimum and maximum over nine measurements.

with matched training windows, target frames, and training budgets. Rollout training increases the mean success of SALT from 84.4% to 92.7%, whereas the LeWM mean decreases from 82.7% to 77.8%. The full configuration achieves the highest mean, indicating that the benefit of recursive rollout supervision depends on the transition structure.

4.4 Planning Reliability: Failure Anatomy on OGBench-Cube

The largest planning gap occurs on OGBench-Cube. We examine a separate paired evaluation of 150 episodes per model, with identical initial states and goals, as described in Section D.

Table 3: **Transition structure and training objective.** Mean closed-loop success rate (%) over Two-Room, Reacher, PushT, and OGBench-Cube, with equal weight assigned to each environment. Training windows, target frames, and training budgets are matched across objectives.

Objective	LeWM	SALT
One-step	82.7	84.4
Rollout	77.8	**92.7**
Δ	**-4.9**	**+8.3**

Mismatch Failures and Unflagged Failures. For each failed episode, we compare two model-predicted terminal costs: $\hat{c}$ is the mean elite cost at the final CEM iteration before execution, and c is the mean elite cost at the first CEM iteration when replanning after execution and re-observation. Both are computed by the same model, at different planning calls and optimization stages, as detailed in Section D. We classify failures using $r = c/\hat{c}$: those with $r > 5$ are *mismatch failures*, and those with $r \leq 5$ are *unflagged failures*. The threshold is selected within a gap in LeWM's seed-1 ratio distribution and held fixed for both models and the remaining seeds. Of the 52 LeWM failures, 35 are mismatch failures.

Mismatch Failures Are Sharply Reduced. Across all 150 paired episodes per model, the mismatch failure rate decreases from 23.3% for LeWM to 2.0% for SALT, while the unflagged failure rate decreases from 11.3% to 4.0%, as shown in Figure 3. Most of the reduction in failures therefore occurs in the mismatch category. Across the 150 paired episodes, SALT succeeds on 43 episodes where LeWM fails, and LeWM succeeds on no episode where SALT fails. These results connect the planning gain to fewer failures in which costs rise sharply after the model observes the execution outcome.

4.5 Long-Horizon Stress Test

We test whether the planning advantage persists at horizons $H \in \{10, 15, 20\}$, executing the full planned sequence before each new observation. This stress test uses goals sampled 100 environment steps after the initial state and an interaction budget of 300 environment steps. The full protocol is given in Section E. As shown in Figure 4, SALT achieves higher success in all twelve combinations of environment and horizon. At $H = 20$, its advantage is 11, 8, 4, and 17 percentage points on Two-Room, Reacher, PushT, and OGBench-Cube, respectively. SALT therefore retains its planning advantage when the transition is applied recursively over longer intervals without new observations.



Figure 3: **Failure anatomy on OGBench-Cube.** Among failed episodes, *mismatch failures* satisfy $r = c/\hat{c} > 5$, while *unflagged failures* satisfy $r \leq 5$. Here, $\hat{c}$ and c are model-predicted terminal costs before execution and during subsequent replanning, respectively. The examples are reproduced LeWM failures, and all images are real observations. Bars report counts over 150 paired episodes per model.

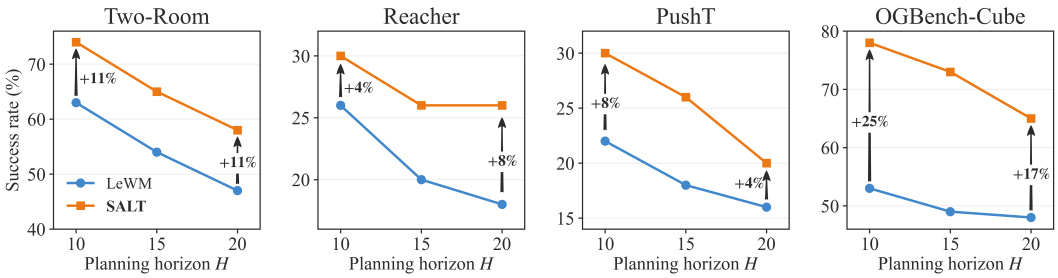


Figure 4: **Long-horizon planning performance.** Success rate on the four environments at $H \in \{10, 15, 20\}$. Arrows mark the performance difference between SALT and LeWM at $H = 10$ and $H = 20$.

4.6 Planning with Multiple Objects

We next test whether the planning advantage extends to goals involving multiple objects. Cube-Double (Park et al., 2025) uses the same robot arm and action space as OGBench-Cube but requires two cubes to reach their target poses simultaneously. Both models are trained using the same protocol as for OGBench-Cube, with three independent runs and 50 evaluation episodes per run.

SALT achieves a success rate of 54.0%, compared with 36.0% for LeWM and 32.7% for the random policy. The 18.0 percentage point gain over LeWM shows that the planning advantage extends to a task requiring both object goals to be satisfied within the same episode.

Table 4: **Success rate (%) on Cube-Double.**

Method	Success rate
Random	32.7±4.2
LeWM	36.0±5.3
SALT	**54.0**±3.5

5 Conclusion

We studied why one-step prediction accuracy can fail to reflect closed-loop planning quality. Our analysis separates errors introduced at individual transitions from their subsequent propagation and characterizes state-affine dynamics by state-independent Jacobians. SALT combines this structure with action conditioning and recursive rollout training. Across four visual planning environments, it achieves higher mean closed-loop success than reproduced LeWM despite larger one-step prediction error. The measured propagation operators exhibit milder amplification over longer rollouts. On OGBench-Cube, SALT reduces failures characterized by sharply higher model-predicted costs after execution and re-observation.

AI USE STATEMENT

Generative AI tools were used to assist with language editing and software development, including code debugging and refinement. All AI-assisted text and code were reviewed and verified by the authors. The authors take full responsibility for the final content of the paper and the correctness of the implementation.

REPRODUCIBILITY STATEMENT

We provide the theoretical derivations and complete proofs in Section A. Implementation details, model parameterization, training configurations, datasets, and evaluation protocols are documented in Section B. Section C–Section F provide the protocols and additional results for the error-propagation analysis, failure analysis, long-horizon evaluation, ablations, and robustness experiments. We will publicly release the source code, training and evaluation scripts, experiment configurations, and trained model weights to support full reproduction of the reported results.

REFERENCES

Kavosh Asadi, Dipendra Misra, Seungchan Kim, and Michel L Littman. Combating the compounding-error problem with a multi-step model. arXiv preprint arXiv:1905.13320, 2019.

Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. In 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 15619–15629. IEEE, 2023.

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.

Randall Balestriero and Yann LeCun. Lejepa: Provable and scalable self-supervised learning without the heuristics, 2025. URL https://arxiv.org/abs/2511.08544.

Steven L Brunton, Bingni W Brunton, Joshua L Proctor, and J Nathan Kutz. Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control. PloS one, 11(2):e0150171, 2016.

Hongyi Chen, ABULIKEMU ABUDUWEILI, Aviral Agrawal, Yunhai Han, Harish Ravichandar, Changliu Liu, and Jeffrey Ichnowski. KOROL: Learning visualizable object feature with koopman operator rollout for manipulation. In 8th Annual Conference on Robot Learning, 2024. URL https://openreview.net/forum?id=A6ikGJRaKL.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In Proceedings of Robotics: Science and Systems (RSS), 2023.

Junha Chun, Youngjoon Jeong, and Taesup Kim. Sparse imagination for efficient visual world model planning. In The Fourteenth International Conference on Learning Representations, 2026. URL https://openreview.net/forum?id=faxcxKINBC.

Yuanlin Duan, Wensen Mao, and He Zhu. Learning world models for unconstrained goal navigation. In The Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024. URL https://openreview.net/forum?id=aYqTwcDlCG.

Christian Gumbsch, Noor Sajid, Georg Martius, and Martin V. Butz. Learning hierarchical world models with adaptive temporal abstractions from discrete latent dynamics. In The Twelfth International Conference on Learning Representations, 2024. URL https://openreview.net/forum?id=TjCDNssXKU.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. Advances in neural information processing systems, 31, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In International conference on machine learning, pp. 2555–2565. PMLR, 2019.

Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=0oabwyZbOu.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. Nature, 640(8059):647–653, 2025.

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

Daning Huang, Muhammad Bayu Prasetyo, Yin Yu, and Junyi Geng. Learning koopman operators with control using bi-level optimization. In 2023 62nd IEEE Conference on Decision and Control (CDC), pp. 2147–2152. IEEE, 2023.

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

Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. In International Conference on Learning Representations, 2017. URL https://openreview.net/forum?id=HyTqHL5xg.

Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. Nature communications, 9(1):4950, 2018.

Lucas Maes, Quentin Le Lidec, Damien Scieur, Yann LeCun, and Randall Balestriero. Leworld-model: Stable end-to-end joint-embedding predictive architecture from pixels, 2026. URL https://arxiv.org/abs/2603.19312.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. In International Conference on Learning Representations (ICLR), 2025.

Reuven Y Rubinstein and Dirk P Kroese. The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning, volume 133. Springer, 2004.

Tankred Saanum, Peter Dayan, and Eric Schulz. Simplifying latent dynamics with softly state-invariant world models. In The Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024. URL https://openreview.net/forum?id=CwNevJONgq.

Haojie Shi and Max Q-H Meng. Deep koopman operator with control for nonlinear systems. IEEE Robotics and Automation Letters, 7(3):7700–7707, 2022.

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

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.

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

Ruixiang Sun, Hongyu Zang, Xin Li, and Riashat Islam. Learning latent dynamic robust representations for world models. In Forty-first International Conference on Machine Learning, 2024. URL https://openreview.net/forum?id=C4jkx6AgWc.

Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning koopman invariant subspaces for dynamic mode decomposition. Advances in neural information processing systems, 30, 2017.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite, 2018. URL https://arxiv.org/abs/1801.00690.

Basile Terver, Tsung-Yen Yang, Jean Ponce, Adrien Bardes, and Yann LeCun. What drives success in physical planning with joint-embedding predictive world models?, 2026. URL https://arxiv.org/abs/2512.24497.

Lucas Thil, Jesse Read, Rim Kaddah, and Guillaume Doquet. Subspace-decomposed jepas: Disentangling progression and content in latent world models, 2026. URL https://arxiv.org/abs/2605.31111.

Lingyi Wang, Rashed Shelim, Walid Saad, and Naren Ramakrishnan. DMWM: Dual-mind world model with long-term imagination. In The Thirty-ninth Annual Conference on Neural Information Processing Systems, 2025a. URL https://openreview.net/forum?id=Bzlt5tPFT6.

Xiyao Wang, Ruijie Zheng, Yanchao Sun, Ruonan Jia, Wichayaporn Wongkamjan, Huazhe Xu, and Furong Huang. COPlanner: Plan to roll out conservatively but to explore optimistically for model-based RL. In The Twelfth International Conference on Learning Representations, 2024. URL https://openreview.net/forum?id=jnFcKjtUPN.

Zizhao Wang, Kaixin Wang, Li Zhao, Peter Stone, and Jiang Bian. Dyn-o: Building structured world models with object-centric representations. In The Thirty-ninth Annual Conference on Neural Information Processing Systems, 2025b. URL https://openreview.net/forum?id=b2u1yrTwFK.

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.

Zhoushun Yu, Xiaoyu Hu, and Xiangyu Xu. Qqworld: Quantile-quantile matching for world model regularization, 2026. URL https://arxiv.org/abs/2607.28415.

Jiahan Zhang, Muqing Jiang, Nanru Dai, TaiMing Lu, Arda Uzunoglu, Shunchi Zhang, Yana Wei, Jiahao Wang, Vishal M. Patel, Paul Pu Liang, Daniel Khashabi, Cheng Peng, Rama Chellappa, Tianmin Shu, Alan Yuille, Yilun Du, and Jieneng Chen. World-in-world: World models in a closed-loop world. In The Fourteenth International Conference on Learning Representations, 2026. URL https://openreview.net/forum?id=yDmb7xAfeb.

Dongdong Zhao, Boyu Li, Fuxiang Lu, Jinhua She, and Shi Yan. Deep bilinear koopman model predictive control for nonlinear dynamical systems. IEEE Transactions on Industrial Electronics, 71(12):16077–16086, 2024.

Kai Zhao, Dongliang Nie, Yuchen Lin, Zhehan Luo, Yixiao Gu, Deng-Ping Fan, and Dan Zeng. Sub-jepa: Subspace gaussian regularization for stable end-to-end world models, 2026. URL https://arxiv.org/abs/2605.09241.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning. In Forty-second International Conference on Machine Learning, 2025. URL https://openreview.net/forum?id=D5RNACOZEI.

A Theoretical Details

A.1 Proof of the Multi-Step Error Decomposition

We use the notation of Section 3.2 and the assumptions of Proposition 1.

Proof. Define the first-order remainder

$$r_t := F(z_{t-1} + \delta_{t-1}, c_{t-1}) - F(z_{t-1}, c_{t-1}) - J_t \delta_{t-1}. \tag{11}$$

Using $\hat{z}_{t-1} = z_{t-1} + \delta_{t-1}$ and adding and subtracting $F(z_{t-1}, c_{t-1})$, we obtain

$$\begin{aligned} \delta_t &= F(z_{t-1} + \delta_{t-1}, c_{t-1}) - F(z_{t-1}, c_{t-1}) + \varepsilon_t \\ &= J_t \delta_{t-1} + r_t + \varepsilon_t. \end{aligned} \tag{12}$$

Since $\delta_0 = 0$, this recursion yields

$$\delta_H = \sum_{j=1}^{H} \Phi_{j \to H}(\varepsilon_j + r_j), \qquad \Phi_{j \to H} = J_H J_{H-1} \cdots J_{j+1}, \qquad \Phi_{H \to H} = I. \tag{13}$$

To verify the expansion, the case $H = 1$ follows from $r_1 = 0$ and $\delta_1 = \varepsilon_1$. If equation 13 holds at $H - 1$, then

$$\begin{aligned} \delta_H &= J_H \sum_{j=1}^{H-1} \Phi_{j \to H-1}(\varepsilon_j + r_j) + (\varepsilon_H + r_H) \\ &= \sum_{j=1}^{H} \Phi_{j \to H}(\varepsilon_j + r_j), \end{aligned} \tag{14}$$

where $J_H \Phi_{j \to H-1} = \Phi_{j \to H}$ and $\Phi_{H \to H} = I$. This proves equation 4.

For the residual bound, assume that $\partial_z F(\cdot, c)$ is L-Lipschitz in the state with the same constant L for every action condition c. Applying the fundamental theorem of calculus along $z_{t-1} + s\delta_{t-1}$ gives

$$r_t = \int_0^1 \big[\partial_z F(z_{t-1} + s\delta_{t-1}, c_{t-1}) - \partial_z F(z_{t-1}, c_{t-1})\big] \delta_{t-1} \, ds. \tag{15}$$

The Lipschitz assumption bounds the norm of the bracketed term by $Ls\|\delta_{t-1}\|$. Therefore,

$$\|r_t\| \le \int_0^1 Ls \|\delta_{t-1}\|^2 \, ds = \frac{L}{2} \|\delta_{t-1}\|^2. \tag{16}$$

□

A.2 Proof of the State-Affine Characterization

Proof of Proposition 2. Fix the action condition c and write $F_c(z) = F(z, c)$.

(Sufficiency.) If $F_c(z) = \mathcal{A}(c)z + \mathcal{B}(c)$, differentiating with respect to z gives

$$\partial_z F_c(z) = \mathcal{A}(c) \qquad \text{for all } z \in \mathbb{R}^d.$$

The Jacobian is therefore independent of the state.

(Necessity.) Suppose $\partial_z F_c(z) = \mathcal{A}(c)$ for all $z \in \mathbb{R}^d$, where $\mathcal{A}(c)$ does not depend on z. Define $G(z) = F_c(z) - \mathcal{A}(c)z$. Then

$$\partial_z G(z) = \partial_z F_c(z) - \mathcal{A}(c) = 0.$$

For any $z, z' \in \mathbb{R}^d$, integration along the segment joining them yields

$$G(z') - G(z) = \int_0^1 \partial_z G\big(z + s(z' - z)\big)(z' - z) \, ds = 0. \tag{17}$$

Thus G is constant on $\mathbb{R}^d$. Denoting this constant by $\mathcal{B}(c)$ gives $F(z, c) = \mathcal{A}(c)z + \mathcal{B}(c)$, completing the proof. □

B Implementation Details

B.1 Environments and Datasets

We follow the LeWM benchmark settings (Maes et al., 2026), including observation and action spaces, success criteria, and training datasets, for Two-Room (2D navigation) (Sobal et al., 2025), Reacher (two-joint reaching) (Tassa et al., 2018), PushT (2D pushing) (Chi et al., 2023; Zhou et al., 2025), and OGBench-Cube (3D robot-arm manipulation) (Park et al., 2025). We split sampled windows into 90% for training and 10% for validation using split seed 3072. Evaluation initial states and goals are sampled from the offline dataset. Cube-Double (Park et al., 2025), described in Section 4.6, follows the OGBench-Cube training and evaluation settings unless stated otherwise.

B.2 Training Configuration

Unless otherwise stated, SALT is trained for 10 epochs with AdamW, a learning rate of 5×10^{-5}, weight decay 10^{-3}, and an epoch-based linear-warmup cosine-annealing schedule. We use batch size 128, `bf16` precision, and gradient clipping at 1.0. The latent dimension is $d = 192$. SIGReg uses weight $\lambda = 0.09$, 17 knots, and 1024 projections.

The visual encoder is a ViT-Tiny trained from scratch, with 12 layers, width 192, 3 attention heads, feedforward dimension 768, and patch size 14. Its CLS-token output is passed through a $192 \to 2048 \to 192$ projection head with batch normalization and GELU to produce the regularized latent state z. The action encoder maps each action block to dimension 192; its input size depends on the frameskip and action dimension. Components and settings shared with reproduced LeWM are listed in Section B.7.

B.3 Transition Model

The transition in equation 8 uses $R = 16$ dense modulation matrices with no low-rank factorization:

$$F(z_t, c_t) = \left[\mathcal{A}_0 + \sum_{r=1}^{R} (W_g c_t)_r N_r \right] z_t + B c_t + b, \qquad c_t \in \mathbb{R}^{192}. \tag{18}$$

It acts directly in the regularized latent space, without the predictor projection head used by LeWM. We parameterize the shared base matrix as

$$\mathcal{A}_0 = U \operatorname{diag}(\tanh s) V^\top, \qquad U = \exp(W_u - W_u^\top), \qquad V = \exp(W_v - W_v^\top). \tag{19}$$

The skew-symmetric generators make U and V orthogonal, so the singular values of $\mathcal{A}_0$ are $|\tanh s_i| < 1$ for finite parameters. This constrains the base matrix, not the full action-conditioned matrix $\mathcal{A}(c)$.

Table 5 lists all stored parameters and their initialization. These values give $U = V = I$, $\mathcal{A}_0 \approx 0.9951 I$, and an initial prediction $F(z_t, c_t) \approx 0.9951 z_t$. The gate W_g has no bias.

Table 5: **Parameters of the SALT transition model** ($d = 192$, $R = 16$). Counts are stored parameters. Only the skew-symmetric parts of W_u and W_v affect the transition, corresponding to $18{,}336$ independent entries for each matrix.

Component	Symbol	Shape	Parameters	Initialization
Generator of U	W_u	192×192	36,864	0
Generator of V	W_v	192×192	36,864	0
Singular-value parameters	s	192	192	3.0
Gate	W_g	16×192	3,072	$\mathcal{U}(-1/\sqrt{192},\, 1/\sqrt{192})$
Modulation modes	$N_1, \ldots, N_{16}$	$16 \times 192 \times 192$	589,824	0
Action injection	B	192×192	36,864	0
Bias	b	192	192	0
Total			**703,872**	

Algorithm 1 Mini-batch training objective for SALT.

```
def affine_step(z, c):
    # z: (B, D), c: (B, D)
    U = expm(W_u - W_u.T)
    V = expm(W_v - W_v.T)

    # Shared base transition: sigma_max(A0) < 1
    A0 = U @ diag(tanh(s)) @ V.T

    # Action-conditioned transition
    g   = c @ W_g.T
    A_c = A0 + einsum('br,rij->bij', g, N)

    return einsum('bij,bj->bi', A_c, z) \
           + c @ B.T + b


def loss(obs, act, lam=0.09):
    # obs: (B, K+1, C, H, W)
    # act: (B, K, ...), action blocks

    # Encode all frames for initialization, targets, and SIGReg
    z = encoder(obs)              # (B, K+1, D)
    c = action_encoder(act)       # (B, K, D)

    # Recursive rollout initialized from the first true latent
    z_cur = z[:, 0]
    preds = []

    for k in range(K):
        z_cur = affine_step(z_cur, c[:, k])
        preds.append(z_cur)

    z_hat = stack(preds, dim=1)     # (B, K, D)

    # Targets are not detached
    loss_roll = mse(z_hat, z[:, 1:])
    loss_reg  = sigreg(z)

    return loss_roll + lam * loss_reg
```

B.4 Training Procedure

Algorithm 1 summarizes the recursive training procedure. A window contains six observations sampled every five environment steps and therefore spans 25 environment steps. Each intervening action block concatenates five raw actions. The first latent initializes the rollout, and all $K = 5$ predictions receive equal supervision. SIGReg uses all six encoded frames, and target latents are not detached.

B.5 Planner and Evaluation Protocol

We use the planning objective and default evaluation protocol in Sections 3.1 and 4.1. CEM uses 300 candidate action sequences per iteration, 30 optimization iterations, 30 elites, and initial sampling variance 1.0. Each model action spans five environment steps, so the default horizon and replanning interval $H = 5$ correspond to 25 environment steps.

The evaluation interface retains LeWM's three-frame history window; SALT uses only the most recent latent state. Each evaluation seed specifies 50 episodes. Matched seeds give paired models identical initial states and goals. The random policy is averaged over three runs.

B.6 Efficiency Measurement

All timings in Table 2 are measured consecutively on the same NVIDIA H100 80 GB GPU. Single-step timing uses a batch of 300 latent states and covers the predictor and its projection head where present. Planning time measures one steady-state batched CEM solve; the reported summary is the unweighted mean over the four environments and $H \in \{5, 10, 15, 20\}$. Training throughput is measured separately for the one-step and rollout configurations.

B.7 Baseline Provenance

Random, reproduced LeWM, and SALT are evaluated in our pipeline. LeWM and SALT are trained in our implementation with matched visual encoders, encoder projection heads, action encoders, latent dimensions, optimizer settings, batch sizes, and CEM planners. LeWM retains its Transformer predictor, predictor projection head, one-step prediction objective, and SIGReg. All paired comparisons use this reproduction.

The PLDM (Sobal et al., 2026), DINO-WM (Zhou et al., 2025), Sub-JEPA (Zhao et al., 2026), SD-JEPA (Thil et al., 2026), and SMWM (Ivashkov et al., 2026) rows in Table 1 are taken from their respective papers. Their training, hyperparameter selection, and reporting protocols differ, so these rows provide context rather than controlled comparisons.

C Empirical Validation of Error Propagation

C.1 Measurement Protocol

The PushT diagnostics in Sections C.2 to C.4 use samples drawn from the training dataset with a fixed generator seed. Rollouts start from an encoded latent state, so $\delta_0 = 0$. Jacobians and matrix products are computed in float64.

We follow the notation of Section 3.2. For SALT, the propagation operator is constructed directly from the learned transition matrices $\mathcal{A}(c)$. For reproduced LeWM, the predictor is conditioned on a three-frame history window. We differentiate only with respect to the most recent latent-state channel while holding the earlier history frames fixed. The resulting $d \times d$ Jacobian therefore does not represent the complete augmented-state Jacobian of the history-based predictor.

C.2 Rollout Reconstruction

We reconstruct the terminal rollout error using the first-order propagation terms

$$\tilde{\delta}_H = \sum_{j=1}^{H} \Phi_{j \to H}\, \varepsilon_j, \qquad \rho_H = \frac{\|\delta_H - \tilde{\delta}_H\|}{\|\delta_H\|}. \tag{20}$$

For SALT, the state-affine transition satisfies the single-state decomposition derived in Section 3.3, so $\rho_H = 0$ in exact arithmetic; nonzero values therefore reflect finite numerical precision.

For reproduced LeWM, only the current-state channel is linearized while the earlier history channels are held fixed. Consequently, ρ_H may contain both nonlinear current-state effects and contributions propagated through the history channels. It should therefore not be interpreted as the nonlinear Taylor remainder alone.

Across $H \in \{1, \ldots, 20\}$, the SALT discrepancy remains at the 10^{-6} level. The corresponding discrepancy for reproduced LeWM increases with rollout horizon and reaches 49.4% at $H = 20$.

C.3 Propagation-Operator Growth

We measure the spectral norm of the propagation operator from the initial state to rollout step k on PushT:

$$\Phi_{0 \to k} = J_k J_{k-1} \cdots J_1. \tag{21}$$

Measurements are aggregated over nine window–batch combinations obtained from three trajectory-window lengths $\{20, 25, 30\}$ and three batches. Each SALT batch contains 40 trajectories. Each LeWM batch contains 16 trajectories because Jacobian computation is substantially more expensive. The reported center is the geometric mean, and the uncertainty band spans the minimum and maximum across the nine measurements.

Between rollout steps 1 and 20, the geometric-mean norm changes from 2.63 to 11.26 for SALT and from 2.46 to 21.51 for reproduced LeWM. These endpoint values underlie the growth factors reported in Section 4.3.

The spectral norm is a worst-case amplification measure. Step-wise operator norms should therefore not be multiplied to estimate the realized amplification of an individual prediction error.

C.4 Control for the Spectral Constraint

To test whether the constraint $\sigma_{\max}(\mathcal{A}_0) < 1$ is responsible for the observed propagation behavior, we train an otherwise identical state-affine model in which $\mathcal{A}_0$ is represented directly as an unconstrained dense matrix initialized to $0.9951I$. All remaining parameters and training settings are unchanged. We then repeat the diagnostic in Section C.3.

Removing the constraint changes the measured twenty-step propagation growth from $4.28\times$ to $4.42\times$, compared with $8.74\times$ for reproduced LeWM.

These measurements do not imply that SALT rollouts are globally contractive.

C.5 Decoded Open-Loop Rollouts

For each environment–checkpoint pair, we freeze the world model and train a post-hoc decoder $D : \mathbb{R}^{192} \to \mathbb{R}^{3\times224\times224}$ on encoded ground-truth observations and their corresponding images. It uses 196 learnable query tokens on a 14×14 grid, four cross-attention blocks with width 512 and eight heads, and a linear output projection to 16×16 RGB patches. Training uses pixel MSE for $30{,}000$ steps with the world model's image preprocessing. The decoder is used only for visualization, never for world-model training, planning, or performance measurement.

Figure 5 uses the SALT checkpoint for each environment and the corresponding dataset actions. The displayed predictions retain recognizable agent, arm, or object configurations across the rollout. They illustrate the observation-space content of predicted latents; quantitative conclusions rely on aggregate measurements rather than these examples.

C.6 Rollout Error and Propagation Across Four Environments

We additionally evaluate rollout error and propagation-operator norms on Two-Room, Reacher, PushT, and OGBench-Cube using test seeds 1, 2, and 3. These measurements complement the nine-measurement PushT summary in Section C.3. As shown in Figure 6, SALT has larger initial prediction error in all four environments, while its measured propagation norms are generally lower at intermediate and later rollout steps. On Reacher, the SALT norm decreases after an initial rise. On PushT, it continues to grow but more slowly than that of LeWM.

Smaller propagation norms do not imply smaller rollout errors at every horizon. On OGBench-Cube, the rollout errors remain close over much of the horizon, and SALT does not have lower error at the final steps shown. The four-environment results therefore broaden the propagation diagnostic beyond PushT while preserving the distinction between error amplification and realized rollout error.

D Failure Anatomy on OGBench-Cube

D.1 Paired Evaluation and Failure Criterion

This diagnostic is separate from the aggregate evaluation in Table 1. Under the default protocol in Section B.5, we evaluate seeds 1–3 with 50 identical episodes per seed for both models, giving 150 paired episodes.

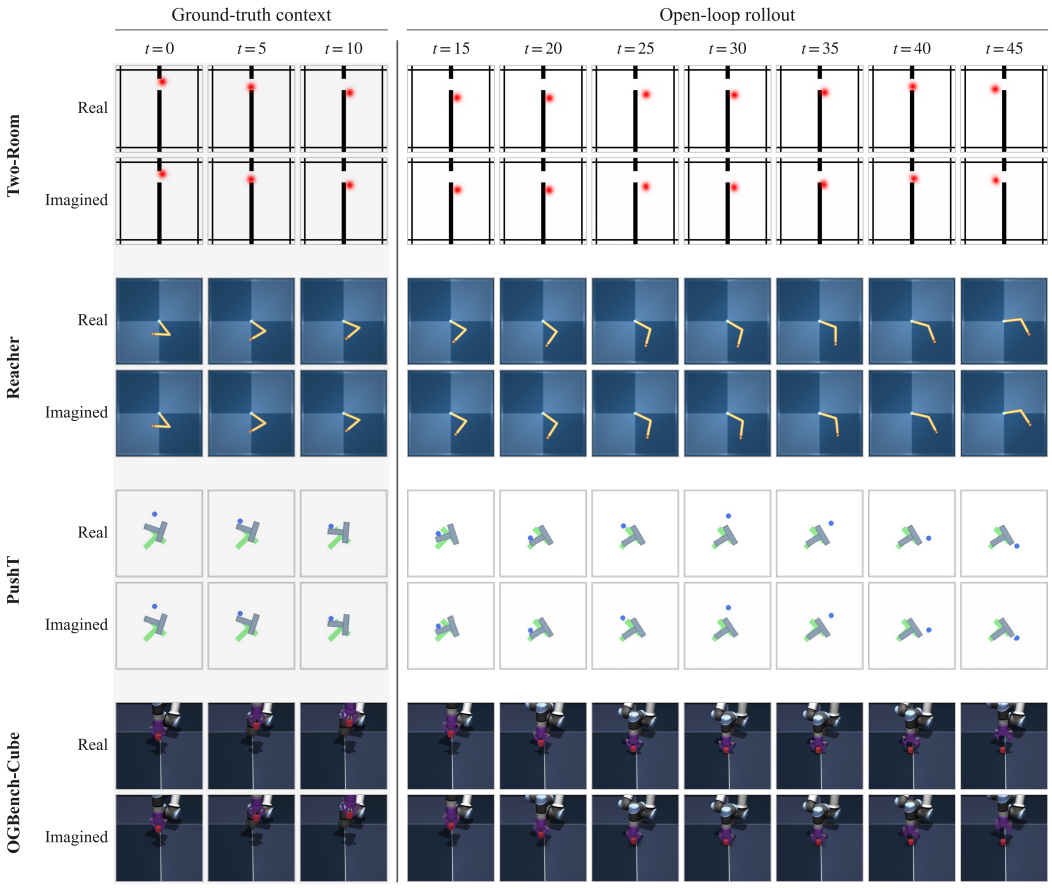


Figure 5: **Decoded open-loop rollouts across four environments.** The first three columns ($t = 0, 5, 10$) are ground-truth context frames from the common evaluation interface. Only the final frame initializes SALT, which predicts seven subsequent latent states using dataset actions. At frameskip 5, these predictions cover 35 additional environment steps ($t = 15, \ldots, 45$). Images are produced by a post-hoc decoder trained only on ground-truth latents and excluded from world-model training and planning.

For each failed episode, $\hat{c}$ is the mean elite cost at the final CEM iteration of the first planning call. Following Section 4.4, we denote by c the mean elite cost at the first CEM iteration of the second call, after executing the first sequence and encoding the resulting observation. Thus c is a post-execution replanning cost; both quantities are computed in the same model's latent space at different planning calls and optimization stages. Their ratio $r = c/\hat{c}$ is not a direct comparison with a physical ground-truth cost.

We select $r = 5$ within the gap in LeWM's seed-1 ratio distribution and keep it fixed for both models and the remaining seeds. Failed episodes with $r > 5$ are *mismatch* failures; those with $r \leq 5$ are *unflagged* failures.

D.2 Failure Composition

Table 6 partitions the 52 LeWM failures and 9 SALT failures. Rates use all 150 episodes per model, not only failed episodes.

Across the paired episodes, both models succeed on 98, only SALT succeeds on 43, only LeWM succeeds on none, and neither succeeds on 9. These counts give success rates of 94.0% for SALT and 65.3% for LeWM in this evaluation, as reported in Section 4.4.

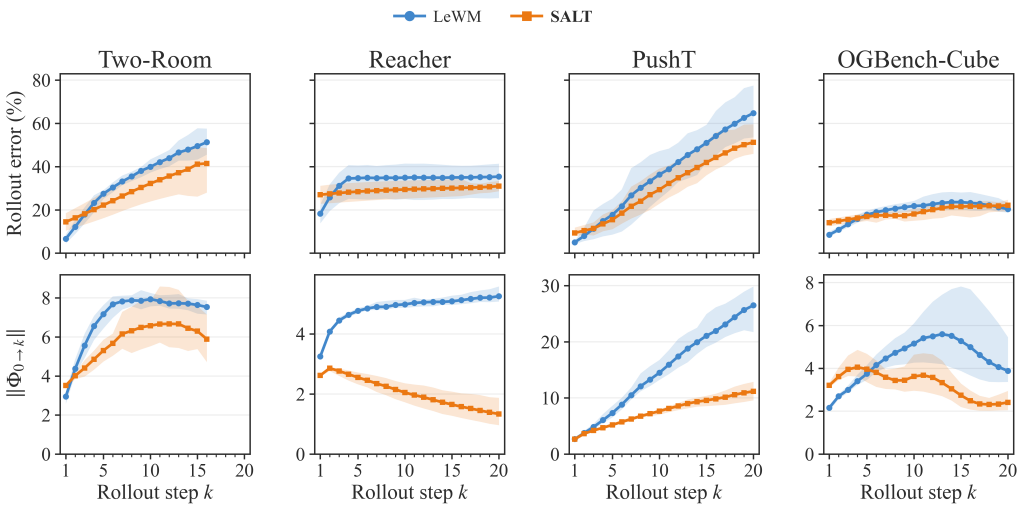


Figure 6: **Rollout error and propagation across four environments.** Top: relative rollout error as a function of rollout step k. Bottom: the spectral norm of the measured propagation operator $\Phi_{0\to k}$. Results use test seeds 1, 2, and 3. Blue circles denote reproduced LeWM; orange squares denote SALT. Each model step corresponds to five environment steps.

Table 6: **Episode-level failure rates by type on OGBench-Cube.** Rates are computed over 150 paired episodes per model using the fixed threshold $r = 5$. LeWM denotes the reproduced baseline.

Model	Mismatch	Unflagged
LeWM	35/150 (23.3%)	17/150 (11.3%)
SALT	3/150 (2.0%)	6/150 (4.0%)

D.3 Figure Details

In Figure 3, the start and goal images are dataset observations; the post-execution image is recorded after the first planned sequence. All are real observations. Both representative examples are failures of reproduced LeWM from evaluation seed 1, with ratios near the seed-1 median of their respective failure types.

E Long-Horizon Planning

E.1 Evaluation Protocol

We use the stress-test protocol in Section 4.5: a target offset of 100 environment steps, a 300-step interaction budget, and planning horizons and replanning intervals $H \in \{10, 15, 20\}$. With frameskip 5, the full-sequence execution intervals are 50, 75, and 100 environment steps. Other settings follow Section B.5; LeWM denotes the reproduced baseline.

E.2 Results Across Four Environments

Table 7 lists the success rates underlying Figure 4. SALT exceeds LeWM in all twelve environment–horizon combinations, with gaps between 4 and 25 percentage points.

F Ablations and Robustness

Each ablation compares variants within the same evaluation round. Absolute values across different ablation groups should therefore not be interpreted as directly comparable.

Table 7: **Success rate (%) under the long-horizon protocol.** Results correspond to the long-horizon experiment reported in the main text.

Environment	Horizon	LeWM	SALT	Gap (pp)
Two-Room	10	63	74	**+11**
	15	54	65	**+11**
	20	47	58	**+11**
Reacher	10	26	30	**+4**
	15	20	26	**+6**
	20	18	26	**+8**
PushT	10	22	30	**+8**
	15	18	26	**+8**
	20	16	20	**+4**
OGBench-Cube	10	53	78	**+25**
	15	49	73	**+24**
	20	48	65	**+17**

F.1 Action-Conditioned Transition

We replace $\mathcal{A}(c)$ with the shared base matrix $\mathcal{A}_0$ while retaining $Bc + b$. Actions therefore remain inputs but no longer modulate the state-transition matrix.

Table 8: **Effect of removing the action-dependent modulation of the transition matrix.** Success rate (%) on Reacher over three evaluation seeds.

Configuration	Planning success
Action-independent transition matrix	11.3 ± 3.1
Full structure	90.7 ± 3.1

Removing action-dependent modulation substantially reduces success on Reacher; the additive action term alone does not match the full model in this setting.

F.2 Robustness

F.2.1 Regularizer Weight

We evaluate SALT across SIGReg weights on PushT. As shown in Figure 7, performance remains stable near the default $\lambda = 0.09$ and decreases toward the tested extremes.

F.2.2 Latent Dimensionality

We keep the visual backbone fixed and adjust the projection-head output, action-condition dimension, and transition parameters to match d. As shown in Figure 8, success varies by at most approximately three percentage points in each environment over the tested fourfold range.

G Limitations and Scope

Scope of the model family. We study JEPA-style world models that plan through recursive latent prediction. Architectures with different prediction and control interfaces, including vision-language-action models, are outside this scope.

Scope of the analysis. Our theoretical and empirical analyses address recursive error propagation, one aspect of latent planning; they do not fully explain all factors determining planning performance.

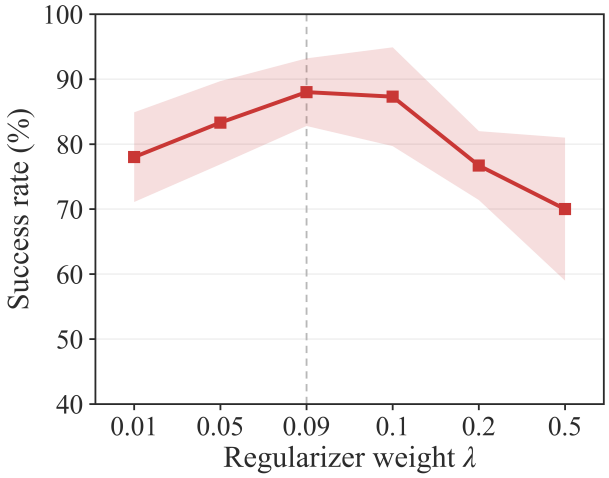


Figure 7: **Success rate against the SIGReg weight on PushT.** Shading shows standard deviation across three evaluation seeds; the dashed line marks the default $\lambda = 0.09$.

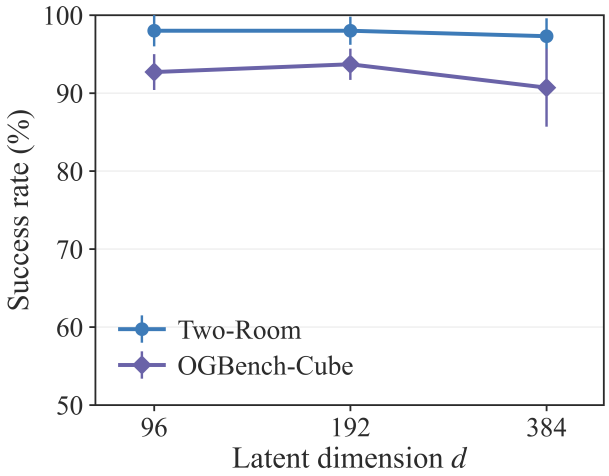


Figure 8: **Success rate of SALT against latent dimension on Two-Room and OGBench-Cube.** The horizontal axis uses a base-two log scale; error bars show standard deviation across three evaluation seeds.