LAFREE WORLD MODEL: LABEL-FREE PRETRAINING FOR ACTION-CONDITIONED JEPA WORLD MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning world models from observation sequences could reduce dependence on synchronized action labels, but learning to explain observed transitions does not necessarily yield dynamics that support downstream control. We introduce LaFreeWM, an end-to-end action-conditioned JEPA world model that learns dynamics using discrete latent actions inferred from consecutive observations. A quantized action interface, together with latent transition objectives, couples action discovery and representation learning without action annotations. A small action-labeled subset subsequently adapts the pretrained world model to the environment's control inputs for planning. Across control benchmarks, LaFreeWM improves planning with limited action supervision while retaining a compact efficient planner. Ablations show the benefit of end to end training and reveal a prediction–planning mismatch: removing quantization substantially lowers held-out latent prediction error while degrading downstream planning. The results demonstrate the value of label-free pretraining for counterfactual planning with action-conditioned world models and highlight the importance of evaluating inferred-action representations through the control they enable.

1 INTRODUCTION

Learning world models from raw sensory observations is a central challenge in building general-purpose agents. A world model that captures how environments evolve under different actions can support planning, policy optimization, and counterfactual reasoning, all without requiring task-specific reward signals during training (Ha and Schmidhuber, 2018; LeCun, 2022). Reconstruction-based approaches such as the Dreamer family (Hafner et al., 2020; 2023) learn latent dynamics models for policy optimization via imagination, but require pixel-level decoding. Joint Embedding Predictive Architectures (JEPAs) provide an attractive alternative: encode observations into compact latent states and predict future embeddings conditioned on actions. This removes the need forpixel-level generation and for reconstructing task-irrelevant details (LeCun, 2022).

Recent JEPA-based world models such as LeWorldModel (LeWM) (Maes et al., 2026) have demonstrated that stable latent world models can be trained end-to-end from pixels using remarkably simple objectives, while achieving fast latent planning competitive with foundation-model-based alternatives like DINO-WM (Zhou et al., 2024). However, all these methods require ground-truth action labels on every training trajectory. This limits scalability to large, cheaply collected video datasets where action labels are absent, partial, or expensive to obtain.

In robotics, action labels come from proprioceptive sensors or teleoperation logs that must be synchronized with camera feeds; in video datasets harvested from the web, no action labels exist at all. A world model that learns action-conditioned dynamics from observation sequences alone, and requires only a small number of action-labeled trajectories to ground them in a given action space, would be substantially more scalable.

A natural solution is to infer actions from data. Inverse dynamics models (IDMs) that predict actions from state-transition pairs have been explored in self-supervised exploration (Pathak et al., 2017) and video-based latent action discovery (Bruce et al., 2024), but integrating them into planning-capable world models introduces new challenges. It creates a shortcut-learning risk (Geirhos et al., 2020; Ye et al., 2023): the conditioning code can transmit target detail that improves observational prediction without supporting control over future transitions. This is because to create the conditioning code

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.

the network has to look at the future state, this is however the state the network also has to predict. The model can thus learn to encode information only of the future state not the transition. This will result achieving low prediction loss but producing codes that are not helping the model to model the environment and making it worse for planning (see our experiments in section 5). Prior work limits the amount of information that is able to be transmitted through a discrete bottleneck (Ye et al., 2023; Bruce et al., 2024); Genie also stops the dynamics loss from updating its latent-action model (Bruce et al., 2024).

In this work, we study how to integrate a self-supervised IDM into an action-conditioned JEPA world model while allowing end-to-end training and learning action codes that support downstream planning. Our approach, LaFreeWM, extends LeWM by inferring actions from pairs of consecutive encoded observations and quantizing them to one of K learned codes before it conditions the dynamics predictor. Quantization limits how much of the next observation a code can carry, regularizing the predictor to model the dynamics of the environment itself.

The prediction loss is backpropagated through the quantized actions so that the encoder, IDM, and predictor are trained jointly. Two auxiliary objectives, a latent-action decoder and a JEPA transition loss, further shape the actions. Once pretraining is complete, a small labeled subset is used to ground the model: the IDM is replaced by direct conditioning on the environment's control inputs, which we call raw actions, and the model is fine-tuned on these labeled trajectories.

Contributions.

- We present LaFreeWM, an end-to-end trained action conditioned JEPA world model without action labels by conditioning its predictor on discrete actions inferred from consecutive observations. A small labeled subset then grounds the world model in the environment's actions for counterfactual planning.
- We isolate a target-information shortcut specific to the coupled IDM–predictor interface: A direct prediction-versus-planning diagnostic shows why low prediction error alone is insufficient for counterfactual control. We also show that the joint training of all components of LaFreeWM greatly improve downstream control.
- We conduct a controlled study where we evaluate LaFreeWM on four continuous control benchmarks and that also evaluates LeWM, PLDM, DINO-WM, Garrido et al. (Garrido et al., 2026) under the same task and evaluation protocol.

2 Related Work

JEPA-style world models. Joint Embedding Predictive Architectures (LeCun, 2022) learn latent representations by predicting embeddings rather than reconstructing pixels. For latent-dynamics planning from offline trajectories, PLDM (Sobal et al., 2025) trains a JEPA-style latent dynamics model with variance-covariance regularization, temporal alignment, and inverse-dynamics losses. DINO-WM (Zhou et al., 2024) sidesteps collapse by freezing a pretrained DINOv2 (Oquab et al., 2024) encoder and learning dynamics over token-level visual features, achieving strong performance at the cost of end-to-end adaptation and substantially higher planning cost. LeWM (Maes et al., 2026) demonstrates that a two-term objective (next-embedding MSE plus SIGReg anti-collapse regularization) suffices for stable end-to-end JEPA training from pixels with a compact model. V-JEPA 2 (Assran et al., 2025) instead action-free-pretrains a video representation before post-training a frozen-encoder, action-conditioned world model for planning, whereas VLA-JEPA (Sun et al., 2026) uses future latents only as supervision targets and deploys an action-prediction policy. Within this landscape, our focus is an observation-inferred action interface inside the end-to-end world model, later grounded in the actions of the environment for planning.

Inverse dynamics and latent action models. Inverse dynamics models (IDMs) that infer actions from state transitions have been used for self-supervised exploration (Pathak et al., 2017), representation learning (Agrawal et al., 2016), and video-based latent action discovery. Genie (Bruce et al., 2024) learns a vector-quantized (VQ) latent action model from video, using a discrete bottleneck to discover controllable action modes without labels. LAPA (Ye et al., 2024) similarly learns discrete pseudo-actions for VLA pretraining before limited-label robot-action fine-tuning, while

LAWM (Tharwat et al., 2025) learns latent actions through world modeling to pretrain imitation models. CLAM (Liang et al., 2025) uses continuous latent actions and jointly trains an action decoder to ease limited-label grounding. MVP-LAM (Lee et al., 2026) uses synchronized cross-viewpoint reconstruction to suppress viewpoint-specific shortcuts, and CLAP (Zhang et al., 2026a) contrastively aligns human-video transitions with quantized robot-action latents. Together, these works use inferred actions for interactive video generation (Genie) or to pretrain and supervise downstream policies (LAPA, LAWM, CLAM, MVP-LAM, and CLAP). In contrast, we use inferred actions to train an action-conditioned world model for counterfactual control. Cross-viewpoint reconstruction and cross-domain alignment remain complementary extensions to our single-view setting.

At larger generative-model scale, DreamDojo (Gao et al., 2026) pretrains continuous proxy actions from human video before robot post-training, DreamGen (Jang et al., 2025) recovers pseudo-actions from world-model-generated robot trajectories for policy training, and DreamZero (Ye et al., 2026) jointly models video and robot actions for closed-loop control. Together, these systems demonstrate the potential of latent-action learning at scale; our work advances this direction by integrating observation-inferred actions into an end-to-end action-conditioned JEPA world model.

The closest planning formulation is Learning Latent Action World Models in the Wild (Garrido et al., 2026), which jointly trains an IDM and predictor on frozen V-JEPA representations and maps known actions into its learned latent interface with a controller. Both approaches target planning; our distinction is joint adaptation of the visual encoder, IDM, and predictor, with a discrete bottleneck. We will show that the interaction between visual encoder and IDM is important which is not possible if the encoder stays frozen. Additionally, we add a novel action shaping loss through a JEPA objective for the latent actions. These different components come together to enhance the IDM for improved latent-action discovery, and codebook utilization for eventual planning performance.

3 METHOD

We describe LaFreeWM, a JEPA-style world model that replaces ground-truth action conditioning with self-supervised latent action codes inferred by an inverse dynamics model (IDM).

3.1 ARCHITECTURE AND LATENT-ACTION LEARNING

Figure 1 summarizes the three stages of LaFreeWM: action-label-free pretraining, limited-label grounding, and planning with raw actions. LaFreeWM uses a Vision Transformer (ViT-Tiny: 12 layers, 3 heads, 192-dim embeddings, patch size 14, Appendix A) as its backbone. The encoder's CLS output is the state embedding $\mathbf{z}_t$; it summarizes the image rather than predicting a class label. The predictor operates on a sequence containing observed state embeddings, with causal attention and action conditioning through adaptive layer normalization (AdaLN).

Instead of encoding ground-truth actions through an action-chunk encoder, LaFreeWM infers discrete latent action codes from consecutive CLS embeddings via an IDM head, passes them through a VQ bottleneck, and uses the resulting quantized codes for prediction conditioning. Figure 1 makes the resulting interface change explicit across the three stages of the method. The future state is available to the IDM only during action-label-free pretraining; limited-label grounding then replaces the IDM–VQ path with raw-action conditioning, so deployment can score proposed actions without access to the future observation.

The IDM is the core component of LaFreeWM. It infers discrete action codes from observed state transitions, providing action conditioning to the dynamics predictor without ground-truth action labels. The IDM of LaFreeWM applies multiple known and novel components which ablations show are necessary during pretraining for eventual controllability. In this section we mention and describe each of these components in detail.

Inverse dynamics head. Given encoder state embeddings $\mathbf{z}_t$ and $\mathbf{z}_{t+1}$ from two consecutive time steps, the IDM maps their ordered pair to a continuous action latent:

$$\mathbf{a}^{(\text{raw})} = f_{\text{IDM}}([\mathbf{z}_t; \mathbf{z}_{t+1}]) \in \mathbb{R}^{d_a}, \tag{1}$$

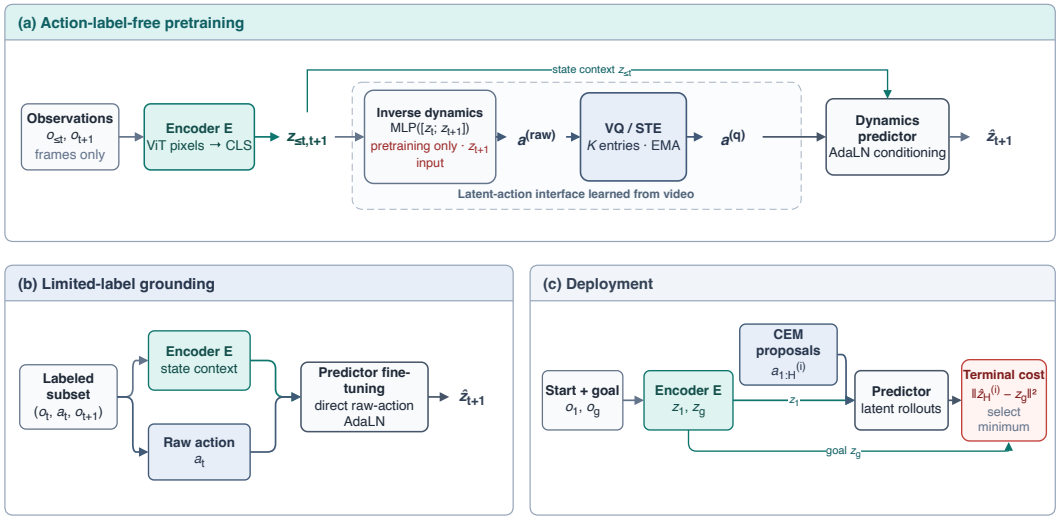


Figure 1: The training and deployment interfaces of LaFreeWM. **(a)** During action-label-free pretraining, the IDM observes consecutive encoded states and supplies a quantized action to the dynamics predictor. **(b)** Limited-label grounding replaces the IDM–VQ interface with direct raw-action conditioning. **(c)** At deployment, CEM proposes raw-action sequences and ranks their predicted terminal states; the future observation and IDM are unavailable.

where $[\cdot;\cdot]$ denotes concatenation, f_{IDM} is the learnable inverse-dynamics mapping, and d_a is the action-latent dimension. Its concrete architecture and output parameterization are specified in Appendix A.2.

VQ bottleneck. The continuous latent passes through a vector-quantization bottleneck (van den Oord et al., 2017) with K learned entries:

$$\mathbf{a}^{(\text{q})} = \mathbf{a}^{(\text{raw})} + \text{sg}\left[\mathbf{e}_k - \mathbf{a}^{(\text{raw})}\right], \qquad k = \arg\min_j \|\mathbf{a}^{(\text{raw})} - \mathbf{e}_j\|_2, \tag{2}$$

Here sg denotes stop-gradient. The model updates codebook entries $\mathbf{e}_j$ by exponential moving averages (EMA) of assigned latents. The straight-through estimator uses the $a^{(\text{raw})}$ gradients, allowing prediction gradients to reach the IDM, while the predictor receives only one of K shared prototypes. For a fixed trained codebook and transition T, the sample-specific channel obeys

$$I(T; \mathbf{a}^{(\text{q})} \mid \mathbf{z}_t) \le H(k \mid \mathbf{z}_t) \le H(k) \le \log_2 K. \tag{3}$$

Equation 3 limits only how much information a single discrete index can transmit through this interface; whether the learned indices capture executable actions is determined by the training objectives and evaluated through planning. Section 5.3 measures marginal code entropy and separately tests code usage and transition content.

JEPA transition shaping. A self-supervised JEPA loss shapes the action latent to predict state transitions. The transition target is the L2-normalized CLS delta:

$$\mathbf{t}_{t\to t+1} = \text{sg}\left[\frac{\mathbf{z}_{t+1} - \mathbf{z}_t}{\|\mathbf{z}_{t+1} - \mathbf{z}_t\|_2}\right]. \tag{4}$$

where sg$[\cdot]$ denotes stop-gradient, and the JEPA loss is the cosine distance between the predicted transition $\hat{\mathbf{t}} = f_\phi(\mathbf{a}^{(\text{q})})$, where f_ϕ is a small predictor MLP, and its target:

$$\mathcal{L}_{\text{JEPA}} = 1 - \cos(\hat{\mathbf{t}}, \mathbf{t}_{t\to t+1}). \tag{5}$$

This provides a self-supervised gradient to the IDM that is independent of action labels: the latent code must capture transition structure, not just prediction-convenient information.

Latent action model (LAM) decoder. A dedicated latent action transition decoder predicts the next state embedding from the current state context and the quantized action code:

$$\hat{\mathbf{z}}_{t+1}^{(\mathrm{LAM})} = \mathrm{LAM}(\mathrm{context}(\mathbf{z}_{\leq t}), \mathbf{a}^{(\mathrm{q})}), \qquad \mathcal{L}_{\mathrm{LAM}} = \left\| \hat{\mathbf{z}}_{t+1}^{(\mathrm{LAM})} - \mathrm{sg}[\mathbf{z}_{t+1}] \right\|_2^2. \tag{6}$$

Here, context summarizes the encoded observations ending at t by pooling tokens, averaging across the history window, and applying a learned projection. $\mathcal{L}_{\mathrm{LAM}}$ Provides a latent-target regression gradient to the IDM through the straight-through estimator(STE). During backpropagation, the straight-through estimator routes the gradient arriving at $\mathbf{a}^{(q)}$ unchanged to the pre-quantized latent $\mathbf{a}^{(\mathrm{raw})}$. The codebook entries themselves remain gradient-free and are updated only by EMA assignments.

3.2 Training objective

The training objective combines the standard JEPA prediction loss with IDM-specific terms:

$$\mathcal{L} = \underbrace{\mathcal{L}_{\mathrm{pred}}}_{\text{next-CLS MSE}} + \lambda_{\mathrm{sigreg}} \underbrace{\mathcal{L}_{\mathrm{SIGReg}}}_{\text{anti-collapse}} + \lambda_{\mathrm{JEPA}} \underbrace{\mathcal{L}_{\mathrm{JEPA}}}_{\text{IDM transition}} + \lambda_{\mathrm{LAM}} \underbrace{\mathcal{L}_{\mathrm{LAM}}}_{\text{action-model decoder}} + \lambda_{\mathrm{VQ}} \underbrace{\mathcal{L}_{\mathrm{commit}}}_{\text{VQ commitment}}. \tag{7}$$

Default loss weights are: $\lambda_{\mathrm{sigreg}} = 0.09$, $\lambda_{\mathrm{JEPA}} = 0.25$, $\lambda_{\mathrm{LAM}} = 1.0$, $\lambda_{\mathrm{VQ}} = 0.25$. SIGReg is the LeJEPA anti-collapse regularizer applied to encoder CLS representations; it matches their empirical characteristic function to an isotropic Gaussian reference (Balestriero and LeCun, 2025). Figure 2 summarizes where each objective attaches: the main prediction loss follows the joint encoder–IDM path, JEPA and LAM use detached transition targets, and the codebook itself is updated by EMA rather than gradients.

Joint gradient flow. In LaFreeWM, the IDM's output is not detached from the main prediction loss. This means the prediction pathway's gradient reaches the IDM head through the straight-through estimator. The main prediction target is also non-detached, so $\mathcal{L}_{\mathrm{pred}}$ reaches the shared encoder. In contrast, the JEPA and LAM targets use stop-gradient, providing additional shaping signal for the latent actions and codebook utilization. Together, these paths let the dynamics predictor and inferred-action representation co-adapt, learning an action conditioned world model with future information without collapsing through shortcut learning.

3.3 Planning and deployment interface

At evaluation time, CEM (Rubinstein, 1999) optimizes raw action sequences against a terminal latent cost. Given initial observation o_1 and goal o_g, the planner selects the action sequence that minimizes $\|\hat{\mathbf{z}}_H - \mathbf{z}_g\|_2^2$.

Because the world model is pre-trained with discrete latent action codes, it cannot directly accept raw actions from the planner. To bridge this gap, we fine-tune the model on a small labeled subset (1% or 10% of the training data with action labels), replacing the VQ-bottlenecked IDM pathway with direct raw-action conditioning. After fine-tuning, CEM proposes raw actions and the predictor evaluates their cost directly, without routing them through the IDM or codebook. The "Labels" column in our results tables indicates the fraction of labeled data used for this fine-tuning step. Thus, action labels are used only for the limited grounding stage rather than for world-model pretraining.

4 Experimental Setup

4.1 Environments

We evaluate on four continuous-control planning benchmarks: **PushT** (Chi et al., 2023) (2D manipulation), **TwoRoom** (grid-world navigation), **Reacher** (DMC (Tassa et al., 2018) continuous control), and **Cube** (OGBench (Park et al., 2024) object manipulation). All environments use offline trajectories of pixel observations (96×96 RGB) and continuous actions collected from pseudo-expert policies. Figure 3 shows example observations from each environment.

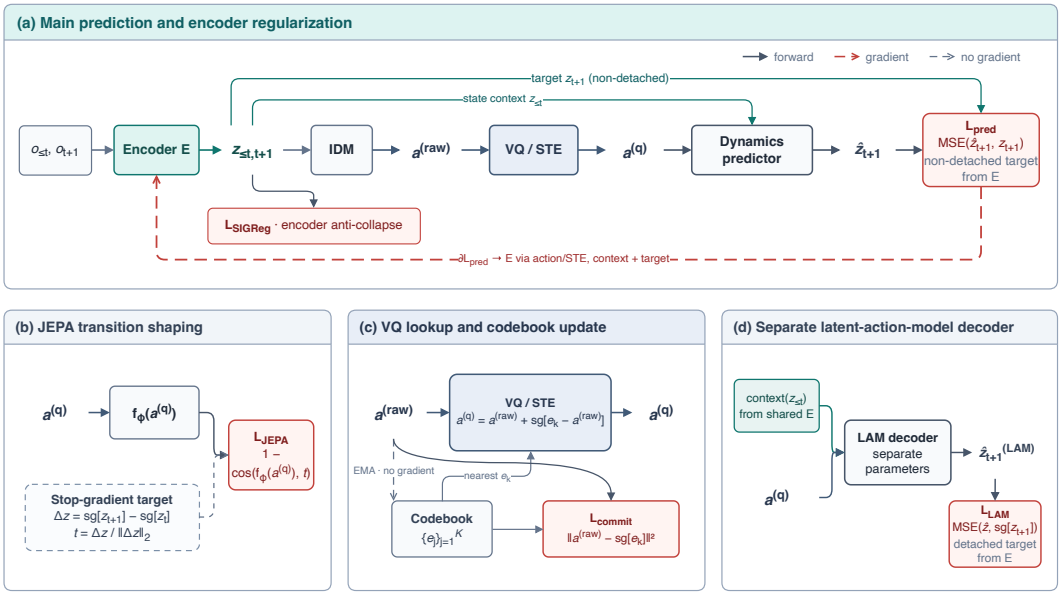


Figure 2: Loss and update paths during pretraining. **(a)** The main prediction loss reaches the shared encoder through the action/straight-through estimator (STE) and SIGReg regularizes encoder CLS states. **(b)** JEPA transition shaping uses a stop-gradient target. **(c)** VQ uses an encoder commitment loss and a non-gradient EMA codebook update. **(d)** The separate LAM decoder uses a detached next-state target.

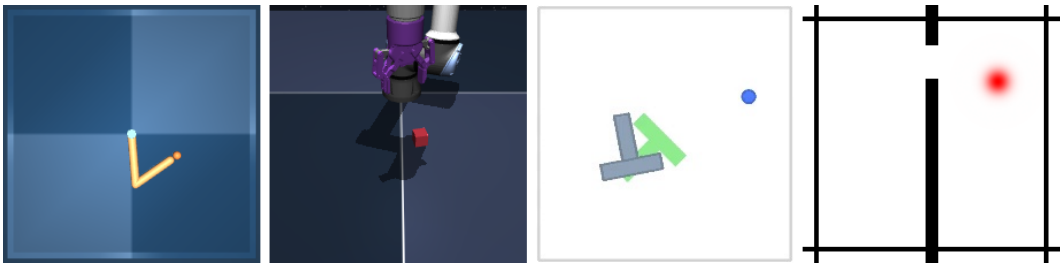


Figure 3: Example observations from the four evaluation environments, ordered left to right: Reacher (DMC continuous control), Cube (OGBench object manipulation), PushT (2D manipulation), and TwoRoom (grid-world navigation).

4.2 Evaluation Protocol

The controlled comparison uses goal horizons $h \in \{1, 5, 10, 15\}$, the implementation uses evaluation budgets $\{5, 10, 20, 30\}$, respectively. The different horizons evaluate the world models on different planning length, with the most important the long horizon planning with $h = 15$. The primary metric is planning success rate (SR, %): the fraction of episodes where the planner reaches the goal within the eval budget. This is measured over 50 episodes. Further details can be found in Appendix B.

4.3 Baselines

Our direct supervision-matched baseline is LeWM (Maes et al., 2026), trained using the 1% or 10% action-labeled episode subset and the same raw-action planning interface. This comparison tests whether action-label-free pretraining improves over learning only from the available labeled trajectories.

We also adapt the noisy continuous latent-action formulation of Garrido et al. (Garrido et al., 2026) to the common simulation and planning interface used by LaFreeWM. It combines a frozen V-JEPA 2-L

encoder with a jointly trained continuous IDM and predictor, followed by an action-to-latent controller trained on the labeled subset.

Fully supervised LeWM, PLDM (Sobal et al., 2025), and DINO-WM (Zhou et al., 2024) provide additional context. PLDM combines latent prediction with variance–covariance regularization, temporal alignment, and inverse-dynamics supervision, whereas DINO-WM plans over token-level features from a frozen DINOv2 encoder. DINO-WM serves as a heavyweight performance reference: under the matched native workload in Appendix C.4, it requires 42.627 seconds per solve compared with 0.287 seconds for LaFreeWM ($148.6\times$ slower).

4.4 Training details

Complete optimization settings, training budgets, hardware, and codebook configurations are reported in Appendix A; the full sensitivity analysis is in Appendix C.2.

5 Results

5.1 Planning performance and efficiency

We evaluate two conditiones of fine-tuning with 1% and 10% of labeled data, and compare it to three fully supervised baselines (trainend on 100% of labeled data) the results are shown in Table 1 . Overall, it shows that LaFreeWM improves control while needing fewer labels and being efficient for planning.

Table 1: Planning success (%, ↑) at $h = 15$. The 100% baselines use all action labels. At 1%/10%, LeWM trains only on the labeled subset, whereas LaFreeWM and Garrido et al. (Noisy) pretrain on trajectories without action annotations before limited-label grounding.

Method	Labels	PushT	TwoRoom	Reacher	Cube
LeWM	100%	30.0 ± 10.7	92.7 ± 2.5	62.7 ± 8.2	77.3 ± 3.4
DINO-WM	100%	86.7 ± 3.8	100.0 ± 0.0	61.3 ± 6.2	74.7 ± 6.2
PLDM	100%	20.7 ± 4.1	68.7 ± 9.0	24.0 ± 10.7	68.7 ± 2.5
LeWM	1%	6.7 ± 0.9	50.7 ± 5.2	20.0 ± 4.3	63.3 ± 7.7
Garrido et al. (Noisy)	1%	13.3 ± 3.4	91.3 ± 5.7	6.7 ± 5.2	60.0 ± 9.9
LaFreeWM	1%	33.3 ± 1.9	76.7 ± 10.4	67.3 ± 7.4	70.7 ± 3.4
LeWM	10%	26.0 ± 5.7	78.0 ± 4.9	60.7 ± 6.2	76.7 ± 2.5
Garrido et al. (Noisy)	10%	16.0 ± 8.2	92.7 ± 5.0	61.3 ± 1.9	71.3 ± 2.5
LaFreeWM	10%	60.7 ± 3.4	83.3 ± 9.6	70.7 ± 5.7	82.0 ± 7.1

At both label budgets, LaFreeWM improves over label-only LeWM in all four environments, showing the benefit of action-label-free pretraining when only a fraction of the trajectories carry action labels. Against Garrido et al. (Noisy), LaFreeWM is higher on PushT, Reacher, and Cube, while the continuous baseline is higher on TwoRoom at both budgets. Subsequent analysis showed that LaFreeWM actually improves in situations where the goal was in the different room, however underperforming in situations when the goal is in the same room. Appendix Table 3 gives the complete horizon profile ($h \in \{1, 5, 10, 15\}$) for all evaluated models.

Planning efficiency. These world models have different sizes and architectures changing their inference times, an important metric for control. We measure end-to-end time per CEM solve (horizon 5, 300 candidates, and 30 iterations) on an H200 GPU. Each method uses its native implementation. LaFreeWM requires 0.287 ± 0.008 s per solve, compared with 0.700 ± 0.012 s for Garrido et al. (Noisy), 2.885 ± 0.086 s for LeWM, 2.996 ± 0.017 s for PLDM, and 42.627 ± 0.164 s for DINO-WM. Appendix C.4 reports the full timing protocol.

5.2 MATCHED IDM ABLATIONS

Table 2: PushT ablations under a common training and deployment interface: planning success (%, ↑). K_{eff} is the effective codebook usage (Section 5.3).

Group	Variant	$h = 1$	$h = 5$	$h = 10$	$h = 15$	K_{eff}
Label construction	Constant action	92	74	16	26.7 ± 4.1	N/A
Component	no VQ	92	64	26	18.0 ± 3.3	N/A
	Without LAM	96	80	20	26.7 ± 1.9	61.24
	Separate IDM	98	76	44	40.7 ± 5.2	3.70
	Gradient-updated VQ	98	88	60	47.3 ± 5.2	10.04
	Detached prediction gradient	100	90	64	56.0 ± 5.9	62.28
	Without JEPA loss	100	98	70	58.0 ± 4.3	44.96
	LaFreeWM, $K = 64$	100	96	70	59.3 ± 8.2	61.53
Codebook	LaFreeWM, $K = 8$	98	94	68	48.7 ± 5.2	7.90
	LaFreeWM, $K = 16$	100	96	78	60.7 ± 3.4	15.68
	LaFreeWM, $K = 128$	96	94	56	51.3 ± 3.4	121.97

Table 2 shows LaFreeWM ($K = 64$, 59.3%) obtains the highest success among the component-ablation variants, and removing or altering any of the coupled pathways reduces planning performance. The separate IDM ablation decouples gradient from the IDM to the encoder. This ablation shows that the encoder benefits from receiving gradients through the IDM, shaping the encoder to produce representations that make it easier for the IDM to produce latent actions that help the dynamics predictor predict the next state. Together, these results support the complete objective, including JEPA transition shaping and the joint gradient path, as a coupled design.

In a separate PushT comparison, using two independently pretrained PushT replicas, No-VQ achieves a next-state feature-prediction error on held-out transitions, i.e. an MSE 1.595×10^{-4} versus 9.588×10^{-3}, about $60\times$ lower, but its planning success is 18.0% versus 60.7%. Thus, the continuous IDM is a better observational predictor yet a worse counterfactual planner. This prediction diagnostic exposes why observational fit is insufficient here.

5.3 WHAT THE LATENT CODES RETAIN

We analyze frozen, action-label-free pretrained models on held-out PushT transitions from offline pseudo-expert trajectories, before raw-action grounding. We measure which codes the IDM assigns, test whether they add information about physical state changes beyond the current state, and replace codes in a frozen predictor to test whether its predictions depend on them. Further details are in Appendix E.

Vocabulary utilization. LaFreeWM with $K = 64$ occupies all 64 entries and has effective code count $K_{\text{eff}} = 2^{H(k)} = 61.53$, with $H(k)$ the entropy measuring how uniform codes are being used effectively. This means the IDM is effectively using 96.1% of the codes (Appendix E). Table 2 reports the matched component variants: removing JEPA lowers the effective count to 44.96, gradient-updated VQ and Separate IDM concentrate assignments into 10.04 and 3.70 effective entries, respectively. This shows that updating the VQ directly, not using EMA collapses the notebook, also removing IDM gradient gradients to the encoder also shows that utilization drops steeply. With both EMA and gradients flowing through the encoder we find that codebook doesn't fully collaps. However, it still not yet used optimally, the additional JEPA loss maximizes the utilization close to the upper limit. Table 2 indicates that utilization and downstream planning complement one another. However, the LAM ablation barely changes utilization ($K_{\text{eff}} = 61.24$), while $h = 15$ planning success falls from 59.3% to 26.7%. The LAM's additional next-state prediction objective helps shape the codes' role, forcing the codes to represent transition information that is more usefull to learn the underlying dynamics of the environment and benefiting eventual planning.

Information about physical transitions. To understand which information a code contains we infer a codes from pairs of consecutive observations. We train probes to predict one-step changes in

the object's physical position and orientation: one receives the current physical state, while the other also receives the inferred code. Adding the code reduces held-out object-displacement RMSE along translation axes and rotation by on average 8.3%. These gains show that the code retains information about the observed transition beyond the current state. This information is then used by the dynamics predictor to model dynamics of the environment. In most environments this even leads to better long horizon planning compared to just training with environment actions.

Use by the dynamics predictor. We hold the observation history and target fixed and replace the code with a different code drawn from a the transition with the most similar physical state based on the distance of environment metrics. This shows if preformance degrades with similar codes. We find that next state embedding prediction error decreases in 90.8% of the cases. Similar results are found when randomly sampling a code. Controlled shifts of the block T or the pusher from a fixed scene produce structured changes in the inferred action latent, indicating sensitivity to local visible change. Across recorded transitions from different scenes, directions remain mixed in the two-dimensional projection. The predictor uses codes as transition-specific cues without requiring each code to have a fixed directional meaning. Appendix E shows further information and corresponding figures.

6 Discussion and Limitations

Label efficiency. The supervision-matched comparison shows that observation-only trajectories provide useful pretraining signal beyond what LeWM can learn from the labeled subset alone. At both the 1% and 10% label budgets, LaFreeWM improves over label-only LeWM in every environment. It also exceeds Garrido et al. (Noisy) on PushT, Reacher, and Cube at both budgets, with TwoRoom as the consistent exception. These results show that action-label-free pretraining with quantised codes can improve downstream planning while using only a fraction of the trajectories with action annotations.

Coupled action and dynamics learning. The ablations support the complete end-to-end objective: JEPA and LAM transition shaping organize inferred actions to contain useful information for dynamics modeling, while the joint gradient path lets the action representation and dynamics predictor co-adapt resulting in better eventual control. The prediction diagnostic shows that this coupling can also be negative if not properly balanced. A continuous IDM can explain observed transitions with much lower prediction error while producing a substantially worse raw-action planner, latent actions have to strike a balance between future information to learn dynamics, but not too much so that the dynamics predictor simply learns to cheat.

Planning efficiency. Under the matched native workload, LaFreeWM completes a solve in 0.287 s compared with 0.700 s for Garrido et al. (Noisy), supporting practical use inside iterative planning. Where as heavy patch level world models such as DINO-WM (42.627 ± 0.164 s) are too computationally intensive for real-time planning.

Limitations. Our evidence is limited to four clean simulation benchmarks, leaving camera motion, embodiment changes, background dynamics, broader behavior distributions, and real-video scale for future evaluation. Raw-action planning still requires a small labeled subset for grounding. The Garrido et al. (Noisy) results use our common-interface simulation adaptation, and native latency reflects each system's implementation and thus these could potentially be further optimized.

7 Conclusion

We introduced LaFreeWM, an action conditioned JEPA world model that learns dynamics without action labels by conditioning on discrete actions inferred from observation transitions, then uses a small labeled subset to ground the predictor for raw-action planning. Across continuous-control tasks, LaFreeWM improves over baselines at a fraction of the labels, while retaining practical planning speed. The prediction–planning diagnostic identifies a central challenge: lower latent prediction error can coincide with substantially worse control when the inferred action carries transition detail that cannot serve as an executable action.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript editing, literature discovery, and preparation of the manuscript in the conference format. AI tools provided feedback on methodological claims, were used for brainstorming and helped with interacting with gpu infrastructure. The authors are responsible for the final text, claims, and artifacts.

ETHICS STATEMENT

This study evaluates world models on simulated control benchmarks. Deploying such models in physical systems requires additional safety validation, particularly when learned dynamics are used to select actions. Broader implications and limitations are discussed in Appendix D.

REPRODUCIBILITY STATEMENT

Section 3 describes the model and objectives. Appendix A specifies the architecture and training configuration, Appendix B describes the evaluation protocol, and Appendix C provides further results and planning-efficiency controls. Code release plans are described in Appendix F.

REFERENCES

Agrawal, P., Nair, A. V., Abbeel, P., Malik, J., and Levine, S. Learning to poke by poking: Experiential learning of intuitive physics. In *NeurIPS*, 2016.

Assran, M., Duval, Q., Misra, I., Bojanowski, P., Vincent, P., Rabbat, M., LeCun, Y., and Ballas, N. Self-supervised learning from images with a joint-embedding predictive architecture. In *CVPR*, 2023.

Assran, M., Bardes, A., Fan, D., Garrido, Q., Howes, R., Komeili, M., Muckley, M., Rizvi, A., Roberts, C., Sinha, K., et al. V-JEPA 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

Balestriero, R. and LeCun, Y. LeJEPA: Provable and scalable self-supervised learning without the heuristics. *arXiv preprint arXiv:2511.08544*, 2025.

Bardes, A., Ponce, J., and LeCun, Y. VICReg: Variance-invariance-covariance regularization for self-supervised learning. In *ICLR*, 2022.

Bardes, A., Garrido, Q., Ponce, J., Chen, X., Rabbat, M., LeCun, Y., Assran, M., and Ballas, N. V-JEPA: Revisiting feature prediction for learning visual representations from video. *arXiv preprint arXiv:2404.08471*, 2024.

Bruce, J., Dennis, M., Edwards, A., Parker-Holder, J., Shi, Y., Hughes, E., Lai, M., Mavalankar, A., Steigerwald, R., Apps, C., et al. Genie: Generative interactive environments. In *ICML*, 2024.

Chi, C., Xu, Z., Feng, S., Cousineau, E., Du, Y., Burchfiel, B., Tedrake, R., and Song, S. Diffusion policy: Visuomotor policy learning via action diffusion. In *RSS*, 2023.

Gao, S., Liang, W., Zheng, K., Malik, A., Ye, S., Yu, S., Tseng, W.-C., Dong, Y., Mo, K., Lin, C.-H., et al. DreamDojo: A generalist robot world model from large-scale human videos. *arXiv preprint arXiv:2602.06949*, 2026.

Garrido, Q., Nagarajan, T., Terver, B., Ballas, N., LeCun, Y., and Rabbat, M. Learning latent action world models in the wild. *arXiv preprint arXiv:2601.05230*, 2026.

Geirhos, R., Jacobsen, J.-H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M., and Wichmann, F. A. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2:665–673, 2020.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. A., Guo, Z. D., Azar, M. G., et al. Bootstrap your own latent: A new approach to self-supervised learning. In *NeurIPS*, 2020.

Ha, D. and Schmidhuber, J. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. In *ICLR*, 2020.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Jang, J., Ye, S., Lin, Z., Xiang, J., Bjorck, J., Fang, Y., Hu, F., Huang, S., Kundalia, K., Lin, Y.-C., et al. DreamGen: Unlocking generalization in robot learning through neural trajectories. *arXiv preprint arXiv:2505.12705*, 2025.

Kingma, D. P. and Welling, M. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

LeCun, Y. A path towards autonomous machine intelligence. *OpenReview*, 2022. Version 0.9.2.

Lee, J. M., Lee, D., Ju, S., Cho, T., Koo, J. W., Zhao, L., Hong, S., and Lee, J. MVP-LAM: Learning action-centric latent action via cross-viewpoint reconstruction. *arXiv preprint arXiv:2602.03668*, 2026.

Liang, A., Czempin, P., Hong, M., Zhou, Y., Biyik, E., and Tu, S. CLAM: Continuous latent action models for robot learning from unlabeled demonstrations. *arXiv preprint arXiv:2505.04999*, 2025.

Maes, L., Le Lidec, Q., Scieur, D., LeCun, Y., and Balestriero, R. LeWorldModel: Stable end-to-end joint-embedding predictive architecture from pixels. *arXiv preprint arXiv:2603.19312*, 2026.

Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., et al. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024.

Park, S., Frans, K., Eysenbach, B., and Levine, S. OGBench: Benchmarking offline goal-conditioned RL. *arXiv preprint arXiv:2410.20092*, 2024.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

Rubinstein, R. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1(2):127–190, 1999.

Sobal, V., Zhang, W., Cho, K., Balestriero, R., Rudner, T. G. J., and LeCun, Y. Learning from reward-free offline data: A case for planning with latent dynamics models. *arXiv preprint arXiv:2502.14819*, 2025.

Sun, J., Zhang, W., Qi, Z., Ren, S., Liu, Z., Zhu, H., Sun, G., Jin, X., and Chen, Z. VLA-JEPA: Enhancing vision-language-action model with latent world model. *arXiv preprint arXiv:2602.10098*, 2026.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., de Las Casas, D., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T., and Riedmiller, M. DeepMind Control Suite. *arXiv preprint arXiv:1801.00690*, 2018.

Tharwat, B., Nasser, Y., Abouzeid, A., and Reid, I. Latent action pretraining through world modeling. *arXiv preprint arXiv:2509.18428*, 2025.

van den Oord, A., Vinyals, O., and Kavukcuoglu, K. Neural discrete representation learning. In *NeurIPS*, 2017.

Ye, S., Jang, J., Jeon, B., Joo, S., Yang, J., Peng, B., Mandlekar, A., Tan, R., Chao, Y.-W., Lin, B. Y., et al. Latent action pretraining from videos. *arXiv preprint arXiv:2410.11758*, 2024.

Ye, S., Ge, Y., Zheng, K., Gao, S., Yu, S., Kurian, G., Indupuru, S., Tan, Y. L., Zhu, C., Xiang, J., et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

Zhang, C., Wang, J., Gao, Z., Su, Y., Dai, T., Zhou, C., Lu, J., and Tang, Y. CLAP: Contrastive latent action pretraining for learning vision-language-action models from human videos. *arXiv preprint arXiv:2601.04061*, 2026a.

Zhang, Z., Wang, Y., Guan, Z., Yang, Y., Shi, B., Zong, T., Yi, H., Chao, G., Chen, X., Yang, T., et al. Delta-JEPA: Learning action-sensitive world models via latent difference decoding. *arXiv preprint arXiv:2606.31232*, 2026b.

Zhou, G., Pan, H., LeCun, Y., and Pinto, L. DINO-WM: World models on pre-trained visual features enable zero-shot planning. *arXiv preprint arXiv:2411.04983*, 2024.

Ye, W., Zhang, Y., Abbeel, P., and Gao, Y. Become a proficient player with limited data through watching pure videos. In *International Conference on Learning Representations (ICLR)*, 2023.

A ARCHITECTURE DETAILS

A.1 BACKBONE CONFIGURATION

LaFreeWM uses a ViT-Tiny backbone with the following configuration:

Parameter	Value
Patch size	14
Embedding dimension	192
Depth (layers)	12
Attention heads	3
Predictor depth	6
Predictor heads	16
Predictor MLP dimension	2048
Predictor dropout	0.1
Action latent dimension	24
History size (PushT, Cube)	3
History size (TwoRoom, Reacher)	1

A.2 INVERSE DYNAMICS HEAD

f_{IDM} consists of an IDM-mode token transformer followed by an output head. It forms a two-token sequence from the source and target 192-dimensional CLS embeddings, adds source/target segment embeddings, and processes the sequence with the 192-dimensional SharedTokenTransformer. The transformer is conditioned through AdaLN by a learned IDM-mode embedding and the horizon embedding; the reported configuration uses 12 layers, 3 attention heads, and an MLP ratio of 4.

The two transformer outputs are flattened and passed through the IDM head: LayerNorm, a $384 \to 192$ affine layer, GELU, and a $192 \to 24$ affine layer, followed by a Tanh output parameterization before quantization. Thus, the "IDM head" refers only to this final projection, whereas f_{IDM} in Eq. 1 includes both the token transformer and this head. The Separate IDM ablation replaces this pathway with a dedicated inverse-dynamics transformer and detaches its state inputs.

A.3 LOSS WEIGHT SUMMARY

Loss term	Weight
Next-CLS prediction MSE	1.0
SIGReg (encoder CLS)	0.09
JEPA transition loss	0.25
LAM decoder loss	1.0
VQ commitment loss	0.25

A.4 TRAINING AND CONFIGURATION DETAILS

All LaFreeWM variants are trained with learning rate 5×10^{-5}, weight decay 10^{-3}, batch size 128, and SIGReg weight 0.09. The component ablations use a VQ codebook size of 64, commitment weight $\lambda_{\mathrm{VQ}} = 0.25$, JEPA weight 0.25, and LAM weight 1.0. The action-label-free world model and fully supervised baselines train for two epochs. Label-only LeWM trains for 10 epochs matching the fine-tuning schedule of LaFreeWM. Table 1 uses different K values per environment as the size of the information bottleneck is optimal for different environments as also shown in Appendix C.2, and Table 3 gives the corresponding multi-horizon results.

A.5 VQ BOTTLENECK DETAILS

The reported Full model updates its codebook entries with EMA. For mini-batch assignments k_i, EMA cluster counts N_j and latent sums $\mathbf{m}_j$ are updated as

$$N_j \leftarrow \gamma N_j + (1-\gamma)\sum_i \mathbf{1}[k_i = j], \qquad \mathbf{m}_j \leftarrow \gamma \mathbf{m}_j + (1-\gamma)\sum_i \mathbf{1}[k_i = j]\mathbf{a}_i^{(\mathrm{raw})}, \tag{8}$$

with $\mathbf{e}_j \leftarrow \mathbf{m}_j / N_j$ (using the implementation's numerical smoothing). The encoder receives the commitment loss

$$\mathcal{L}_{\mathrm{commit}} = \|\mathbf{a}^{(\mathrm{raw})} - \mathrm{sg}[\mathbf{e}_k]\|_2^2, \tag{9}$$

and the straight-through construction in Eq. 2 carries downstream gradients to the IDM encoder. The LaFreeWM implementation uses EMA updates, while the Gradient-updated VQ row in Table 2 evaluates direct codebook optimization.

A.6 ACTION FINE-TUNING DETAILS

During pre-training, the predictor receives the quantized IDM output $\mathbf{a}^{(\mathrm{q})}$. During grounding, we switch the action source from the IDM to the observed action sequence. The IDM, VQ assignments, and codebook are therefore bypassed during this stage.

Raw actions are first normalized using the training-set action statistics and mapped to per-step conditioning vectors by the LeWM-compatible action embedder. This embedder consists of a 1×1 convolution mapping the action vector to 10 intermediate channels, followed by an MLP with hidden width $4d_z$ and output dimension $d_z = 192$, and a final LayerNorm. The resulting vectors $\mathbf{c}_{t+i}$ condition the corresponding predictor positions through adaptive layer normalization. Grounding minimizes the dense next-state CLS prediction loss over the labeled transitions. The visual encoder, action embedder, predictor, and their projection layers are optimized jointly, while the IDM-specific objectives are inactive because the predictor is conditioned directly on raw actions.

The reported 1% and 10% label regimes refer to episode-level labeled subsets. For the matched PushT component study, the 10% checkpoint is obtained through a cumulative $1\% \rightarrow 3\% \rightarrow 5\% \rightarrow 10\%$ ladder. Each stage is initialized from the preceding stage and uses the same optimizer, batch size, learning rate, weight decay, and ten-epoch training schedule for every variant. The 10% stage uses AdamW with learning rate 5×10^{-5}, weight decay 10^{-3}, batch size 128, and gradient clipping at 1.0.

B EVALUATION CONFIGURATION DETAILS

All reported single-step experiments use dataset frameskip 1 and one primitive action per control step. Observations are stored as 96×96 RGB images and resized to 224×224 before entering the ViT backbone. PushT and Cube use an observation history of three frames, whereas TwoRoom and Reacher use a history of one frame.

We evaluate goal horizons $h \in \{1, 5, 10, 15\}$. The goal observation is selected at temporal offset h from the initial observation, while the planner is allowed $\{5, 10, 20, 30\}$ primitive action steps for these four horizons, respectively. Thus, the goal horizon specifies how the target is constructed, whereas the evaluation budget specifies how many actions the planner may execute.

For every model, horizon, and evaluation seed, we evaluate 50 fixed episode–goal instances. The $h = 15$ results are evaluated independently with seeds 0, 1, and 2 and report the arithmetic mean and population standard deviation of the three per-seed success rates. The shorter-horizon results use seed 0 only.

Success is cumulative within the evaluation budget: an episode is counted as successful if the goal is reached at any point before the budget is exhausted. The planner uses single-step action blocks; detailed CEM workload settings are reported in Appendix C.4.

C ADDITIONAL RESULTS

C.1 MULTI-HORIZON RESULTS ACROSS ENVIRONMENTS

Table 3 reports success rates at goal horizons $h \in \{1, 5, 10, 15\}$ across all four environments under the single-step regime. The methods and configurations match Table 1; the h=15 values reproduce that comparison, while h<15 values are seed-0 point estimates.

Table 3: Multi-horizon planning success (%, ↑) under frame-skip 1. $h = 15$: mean ± population std over three evaluation seeds; $h < 15$: seed-0 point estimates.

Env	Method	Labels	h=1	h=5	h=10	h=15
PushT	LeWM	100%	88	62	50	30.0 ± 10.7
	DINO-WM	100%	100	100	100	86.7 ± 3.8
	PLDM	100%	98	60	28	20.7 ± 4.1
	LeWM	1%	26	30	20	6.7 ± 0.9
	Garrido et al. (Noisy)	1%	54	38	16	13.3 ± 3.4
	LaFreeWM	1%	96	60	36	33.3 ± 1.9
	LeWM	10%	86	64	42	26.0 ± 5.7
	Garrido et al. (Noisy)	10%	74	50	32	16.0 ± 8.2
	LaFreeWM	10%	100	96	78	60.7 ± 3.4
TwoRoom	LeWM	100%	100	100	100	92.7 ± 2.5
	DINO-WM	100%	100	100	100	100.0 ± 0.0
	PLDM	100%	100	94	74	68.7 ± 9.0
	LeWM	1%	100	96	64	50.7 ± 5.2
	Garrido et al. (Noisy)	1%	100	98	94	91.3 ± 5.7
	LaFreeWM	1%	100	100	98	76.7 ± 10.4
	LeWM	10%	100	98	94	78.0 ± 4.9
	Garrido et al. (Noisy)	10%	100	100	96	92.7 ± 5.0
	LaFreeWM	10%	100	100	92	83.3 ± 9.6
Reacher	LeWM	100%	80	34	56	62.7 ± 8.2
	DINO-WM	100%	98	62	64	61.3 ± 6.2
	PLDM	100%	66	24	26	24.0 ± 10.7
	LeWM	1%	54	22	26	20.0 ± 4.3
	Garrido et al. (Noisy)	1%	26	26	18	6.7 ± 5.2
	LaFreeWM	1%	64	64	76	67.3 ± 7.4
	LeWM	10%	76	40	46	60.7 ± 6.2
	Garrido et al. (Noisy)	10%	62	44	74	61.3 ± 1.9
	LaFreeWM	10%	68	68	82	70.7 ± 5.7
Cube	LeWM	100%	100	100	88	77.3 ± 3.4
	DINO-WM	100%	100	96	88	74.7 ± 6.2
	PLDM	100%	100	94	76	68.7 ± 2.5
	LeWM	1%	100	80	76	63.3 ± 7.7
	Garrido et al. (Noisy)	1%	100	94	94	60.0 ± 9.9
	LaFreeWM	1%	100	98	86	70.7 ± 3.4
	LeWM	10%	100	96	84	76.7 ± 2.5
	Garrido et al. (Noisy)	10%	100	92	94	71.3 ± 2.5
	LaFreeWM	10%	100	100	94	82.0 ± 7.1

C.2 CODEBOOK SENSITIVITY

Table 4 reports the full 10% codebook sensitivity at $h = 15$. $K = 16$ has the highest average across the four tasks, with $K = 8$ being highest on Reacher and $K = 128$ on Cube.

Table 4: LaFreeWM codebook sensitivity at $h = 15$ (10% labels), mean $\pm$ population std over three evaluation seeds.

Codebook	PushT	TwoRoom	Reacher	Cube
$K = 8$	48.7 ± 5.2	78.0 ± 7.5	70.7 ± 5.7	76.7 ± 5.2
$K = 16$	60.7 ± 3.4	83.3 ± 9.6	66.7 ± 9.0	79.3 ± 6.6
$K = 64$	59.3 ± 8.2	75.3 ± 12.3	64.7 ± 4.7	76.0 ± 5.9
$K = 128$	51.3 ± 3.4	76.7 ± 9.8	64.7 ± 2.5	82.0 ± 7.1

C.3 Conditional pretraining-seed stability

We test sensitivity to pretraining initialization in the PushT $h = 15$ component study with 10% action labels. We repeat four variants over three pretraining seeds: LaFreeWM Full ($K = 64$), Without VQ, Without the LAM decoder, and Detached prediction gradient (which blocks the prediction loss from updating the IDM). For each pretrained model, we hold the fine-tuning seed fixed and evaluate under three planner/environment seeds. Table 5 reports the resulting planning success.

Table 5: PushT $h = 15$ success (%) across three pretraining seeds, with fixed fine-tuning seed.

Variant	Success across pretraining seeds
LaFreeWM	54.4 ± 8.1
Detached prediction gradient	50.9 ± 2.3
Without LAM	29.8 ± 1.9
Without VQ	15.8 ± 2.8

C.4 Native-planner efficiency

Table 6 uses the same PushT CEM workload on H200: horizon 5, 300 candidates, 30 iterations, and top-k 30. The synchronized timer encloses one complete solve and excludes loading, simulator interaction, rendering, logging, and success evaluation.

Table 6: End-to-end native-planner measurements on H200. Latency is mean $\pm$ population std over three per-process medians. Each row uses the method's native precision, caching, and input path.

Planner	Precision	Params.	Latency (s)	Peak GiB	Relative
LaFreeWM	BF16	27.52M	0.287 ± 0.008	0.43	$1.00\times$
Garrido et al. (Noisy)	BF16	376.75M	0.700 ± 0.012	0.85	$2.44\times$
LeWM	FP32	18.03M	2.885 ± 0.086	0.46	$10.06\times$
PLDM	FP32	18.03M	2.996 ± 0.017	0.46	$10.44\times$
DINO-WM	FP32	52.32M	42.627 ± 0.164	17.82	$148.60\times$
DINO-WM cached control	FP16	52.32M	20.126 ± 0.005	15.83	$70.16\times$

D Broader impact

This work contributes to world models for robotic planning. Self-supervised world models could reduce dependence on expensive labeled demonstrations, lowering barriers to deploying robots in new environments. The VQ-bottlenecked IDM design could improve scalability by enabling world-model training on large video datasets without action annotations.

World models that learn from video without action supervision could be applied to settings where autonomous behavior is undesirable or where training data encodes unsafe behaviors. Extending the current 2D and simulated 3D results to real-world deployment requires further validation. The fine-tuning step on labeled data introduces a natural supervision checkpoint: the need for some labels at deployment provides an opportunity for human oversight.

E LATENT-CODE UTILIZATION AND TRANSITION INFORMATION

Effective vocabulary. For held-out assignment frequencies p_j, we report occupied entries and

$$H(k) = -\sum_{j:p_j>0} p_j \log_2 p_j, \qquad K_{\text{eff}} = 2^{H(k)}. \tag{10}$$

K_{eff} equals K for uniform assignments and approaches one when assignments concentrate on a single entry. All six variants in Table 7 use the same 7,476 transitions from 1,869 held-out episodes, the same pretraining seed, and the two-epoch checkpoint stage.

Table 7: Matched pre-grounding PushT vocabulary usage ($K = 64$). The effective count distinguishes broad occupancy from balanced assignment frequencies.

Variant	Occupied	K_{eff}	K_{eff}/K (%)	Top-three mass (%)
LaFreeWM	64	61.53	96.1	6.7
Without LAM	64	61.24	95.7	7.9
Without JEPA	63	44.96	70.3	18.0
Detached prediction gradient	64	62.28	97.3	6.7
Separate IDM	8	3.70	5.8	94.4
Gradient-updated VQ	17	10.04	15.7	45.8
LaFreeWM, $K = 8$	8	7.90	98.7	43.5
LaFreeWM, $K = 16$	16	15.68	98.0	24.2
LaFreeWM, $K = 128$	128	121.97	95.3	4.3

Physical-transition probes. We compare a current-physical-state probe with a state-plus-code probe, using nonlinear estimators. The discrete $K = 16$ code is represented by a one-hot vector; it is inferred from the source and subsequent frame by the frozen IDM. Probe settings are selected on the development split, and targets are one-step physical-state changes. Table 8 reports object components on 7,476 test transitions from 1,869 episodes.

Table 8: Held-out physical-transition probe errors for PushT $K = 16$. Negative paired differences favor adding the code.

Target	State only	State + code	Reduction
Object Δx (pixels)	1.6457	1.5330	6.8%
Object Δy (pixels)	1.8074	1.6815	7.0%
Object $\Delta\theta$ (radians)	0.02567	0.02282	11.1%

Controlled local pose responses. To examine how LaFreeWM visible change, we start from one PushT scene and move either the T-shaped block or the pusher across a grid of horizontal and vertical offsets, leaving the other body fixed. For each edited image, we compare its encoder embedding and the continuous output of the inverse dynamics model (IDM), which receives both the original state and next state. Figure 4 shows the T edits in the top row and the pusher edits in the bottom row; LaFreeWM image embeddings are on the left and LaFreeWM IDM outputs are on the right. In middle it shows the outpus of LeWM We move either the T-shaped block (top row) or the pusher (bottom row) over a 21×21 grid of horizontal and vertical offsets from -40 to $+40$ pixels in 4-pixel steps, keeping the other body and the T's orientation fixed. The left column projects its 192-dimensional image embedding. The right column projects the 24-dimensional continuous inverse-dynamics latent inferred from the original and edited image embeddings, before vector quantization. Color indicates the moved body's horizontal offset.

Recorded transitions across scenes. We next examine the same frozen PushT model on recorded one-step transitions rather than directly edited images. For each pair of consecutive observations, the IDM infers a continuous 24-dimensional latent from the before and after image embeddings. Figure 5 projects the 518 pairs whose RGB images visibly differ and colors the same projected points by inferred discrete code or by the recorded action. The pairs come from 15 selected episode windows.



Figure 4: Local PushT pose responses for LaFreeWM. Starting from one scene, we move either the T-shaped block (top row) or the pusher (bottom row) over a 21×21 grid of horizontal and vertical offsets.

Each point is one of 518 recorded before/after image pairs from 15 selected episode windows, after excluding 506 pairs with identical RGB images from the original 1,024-transition sample. LaFreeWM infers a continuous 24-dimensional inverse-dynamics latent for each pair. We fit one unsupervised UMAP projection to these latents (Euclidean distance, 30 neighbors, minimum distance 0.1); both panels show exactly the same coordinates. Panel (a) colors points by their nearest learned codebook entry. Panel (b) colors the same points by the recorded motor action grouped into eight directions and a still category.

The two figures address different questions. Figure 4 holds the starting scene fixed and deliberately moves one body, revealing how the inferred latent responds to a known visible effect. Figure 5 instead combines recorded transitions with different starting scenes, contacts, and resulting effects. Its code-colored neighborhoods are partly expected because codes are assigned from the plotted continuous latents, while the recorded action categories remain mixed in the same coordinates.

Controlled predictor substitutions. We use 512 valid samples to evaluate LaFreeWM with $K = 64$. The predictor context and target remain identical across arms; only the final quantized action vector changes. Alternative codes are taken from the probe-training split, including a different code from a nearest-current-state donor in another episode, a draw from training code frequencies, and a code matched to the donor replacement's codebook distance. Predictor errors are Euclidean distances to the same target embedding. Table 9 reports the mean paired increase.

The similar-current-state substitution increases error in 465/512 windows (90.8%). The identity checks verify that the observed differences arise from changing the predictor's code input.

F Reproducibility

The implementation release will include the model, training, grounding, planning, and evaluation code together with the exact experiment configurations used in the paper. These configurations record the data split and label manifest, frameskip and history settings, optimization schedule, checkpoint,



Figure 5: Inferred codes and recorded actions for visibly changing PushT transitions.

Table 9: Code substitution in the frozen LaFreeWM $K = 64$ predictor. Intervals use paired episode-cluster resampling. Repeating the original vector gives exactly zero change.

Replacement	Changed windows	Normalized error increase
Similar-current-state donor	512	0.0374
Training-frequency sample	505	0.0347
Codebook-distance matched	512	0.0325
Original vector (identity)	0	0.0000

planner parameters, evaluation budgets, and seeds for each reported result. Code and experiment configurations will be made publicly available with the final paper.