JACT: LEARNING TO ACT THROUGH FROZEN JEPA DYNAMICS

Anonymous authors
Manuscript in preparation

ABSTRACT

Visual world models can guide control through online planning, but repeated prediction and search add deployment cost. We propose JACT (JEPA-guided Action-Consequence Training), which moves the use of predictive dynamics into offline policy training. A direct policy acts recursively inside a frozen joint-embedding predictive architecture (JEPA), conditions subsequent decisions on predicted consequences, and learns to align the resulting trajectory with observed future embeddings. Gradients pass through the fixed dynamics into the actor, while local supervision anchors its actions. Deployment uses only the encoder and policy, with no predictive rollout, search, or retrieval. Across 11 visual control settings, JACT combines broad task coverage with efficient direct execution. It achieves 97.25% average success on the four-task comparison, the highest among the listed methods with complete coverage. Controlled ablations and repeated actor training support the value of recursive consequence supervision, while identifying task-dependent limits. JACT offers a simple route from pretrained JEPA dynamics to direct visual control.

1 INTRODUCTION

Visual world models predict how actions change the world. Joint-embedding predictive architectures (JEPAs) learn through representation-space prediction (Assran et al., 2023); action-conditioned models extend this principle to control without reconstructing pixels (Zhou et al., 2025; Assran et al., 2025; Maes et al., 2026). Online planning exploits these models by repeatedly proposing actions and evaluating imagined outcomes. Can we instead use JEPA dynamics to train a policy that acts directly?

Direct policies provide an efficient alternative. GC-IDM learns goal-conditioned action prediction on frozen embeddings (Nguyen et al., 2026), but leaves the action-conditioned predictor unused. Local action regression supervises decisions at recorded states; it does not directly evaluate how one decision changes the context of the next. A predictor can supply this missing signal about actions and their sequential interaction.

Our central idea is to *train the policy on the consequences of its own decisions inside a frozen JEPA*. The policy generates actions, the predictor propagates their effects, and the predicted context conditions the next decision. Encoded observations supervise these futures at multiple boundaries. Future discrepancies therefore train both the actions that caused them and earlier decisions that shaped subsequent policy inputs.

We realize this idea as JACT (**J**EPA-guided **A**ction-**C**onsequence **T**raining; Figure 1). We start with a supervised waypoint-conditioned actor, denoted JACT (Stage 1) before consequence training and JACT (full) under the complete two-stage procedure. The second stage adds recursive consequence supervision while retaining the local anchor. Gradients pass through frozen dynamics, but only actor weights change. Deployment uses only the encoder and actor, with no predictive rollout, search, or retrieval.

The distinguishing design is a *recursively coupled policy–predictor trajectory*. Unlike forward consistency at recorded contexts (Pathak et al., 2018), later decisions depend on earlier predicted consequences. Unlike reward-driven learning through dynamics (Heess et al., 2015; Hafner et al., 2020),

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

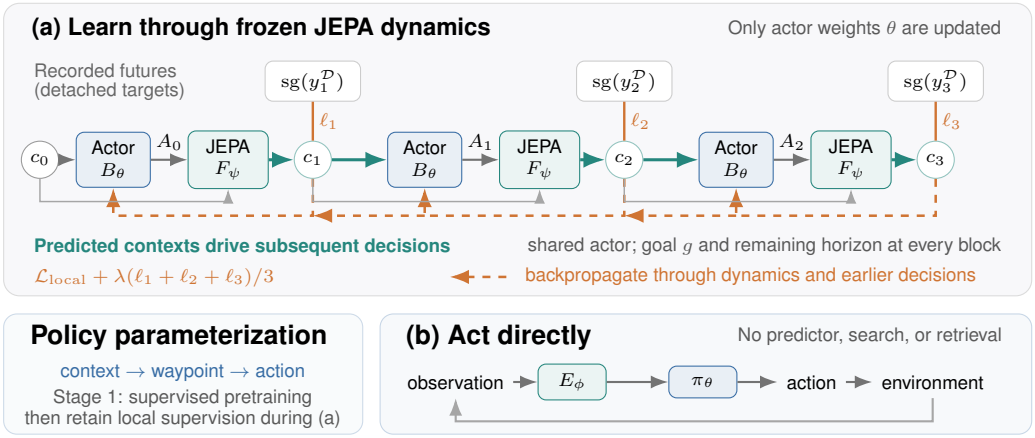

Figure 1: **JACT learns through predicted consequences and acts directly. (a)** The actor acts inside frozen JEPA dynamics. Predicted contexts feed subsequent decisions (teal), while detached encodings of recorded futures supervise their consequences. Orange dashed arrows denote gradients through the frozen dynamics and earlier decisions; only the shared actor weights are updated. Local supervision from Stage 1 is retained at the initial context. **(b)** Deployment uses only the encoder and actor, without predictive rollouts, search, or retrieval. The lower-left inset summarizes the policy instance.

supervision comes from observed offline futures, without reward or value fitting. The waypoint actor instantiates the method; the contribution is how predictive dynamics train its decisions.

Across 11 visual control settings, JACT combines broad coverage with direct execution. It achieves 97.25% average success in the four-task comparison, the highest among listed methods with complete coverage. Controls and repeated actor training support consequence supervision most clearly on Puzzle-4×4 and PointMaze, with no consistent gain on Wall. Further studies quantify deployment cost, actor-data efficiency, and fixed-policy performance across goal horizons.

Our contributions are: (i) **a training-time interface to JEPA dynamics**, using observed futures to supervise direct policies; (ii) **recursive action-consequence training**, coupling successive decisions through predicted contexts while retaining local supervision; and (iii) **evidence for performance and mechanism**, from complete-controller comparisons, same-label and temporal controls, and deployment-cost analysis. Prediction shapes the policy during training without remaining in its deployment graph.

2 Preliminaries

We consider goal-conditioned control from a fixed offline dataset $\mathcal{D}$ of observation–action trajectories. Each training item contains an observed context, a future goal observation o_g, a time-to-go H in environment steps (wrapper steps for Reacher), and intervening actions and observations. A frozen encoder E_ϕ maps observations into latent states, and a frozen action-conditioned JEPA predictor F_ψ models transitions:

$$x_t = E_\phi(o_t), \qquad \hat{x}_{t+q} = F_\psi(c_t, A_t), \quad A_t = (a_t, \ldots, a_{t+q-1}). \tag{1}$$

Here c_t contains the state and action history required by the predictor, and one prediction block spans q environment steps. The full model state may contain visual patches and proprioception. We distinguish its actor-facing readout $z = P(x)$ from its visual supervision readout $Q(x)$; a pooled actor input is not necessarily the full predictive state. The policy goal is $g = P(E_\phi(o_g))$.

One policy call outputs an action block covering b environment steps, which need not equal q. We use $\hat{a}$ for this possibly vector-valued output and A for a full world-model action block. Intermediate future observations supervise training only; deployment receives the current observation and a specified goal image. Our default rollout uses $K = 3$ prediction blocks of $q = 5$ steps each.

3 JEPA-GUIDED ACTION-CONSEQUENCE TRAINING

3.1 LEARNING FROM POLICY-INDUCED CONSEQUENCES

Let B_θ be a differentiable policy that generates one q-step action block from a context, goal, and remaining horizon. We start from a supervised policy and retain its factual-state objective $\mathcal{L}_{\text{local}}$ as an anchor. The additional objective supervises the trajectory induced by that policy. Starting from factual context c_0, we alternate action generation and prediction:

$$A_j = B_\theta(c_j, g, H - jq), \qquad j = 0, \dots, K-1, \tag{2}$$

$$\hat{x}_{j+1} = F_\psi(c_j, A_j), \qquad c_{j+1} = U(c_j, A_j, \hat{x}_{j+1}), \tag{3}$$

where U appends the predicted state and actions while preserving the model's required history. Crucially, the next actor input comes from the predicted context, not the recorded future.

For each boundary, let $y^{\mathcal{D}}_{j+1} = Q(E_\phi(o_{t+(j+1)q}))$ be the visual representation of the actual future observation. We minimize the normalized discrepancy between the policy-induced trajectory and these observed targets:

$$\mathcal{L}^{(K)}_{\text{con}} = \frac{1}{K} \sum_{j=0}^{K-1} \text{mean} \left[\left(\frac{Q(\hat{x}_{j+1}) - \text{sg}(y^{\mathcal{D}}_{j+1})}{S} \right)^2 \right], \tag{4}$$

$$\min_\theta \; \mathcal{L}(\theta) = \mathbb{E}_{\mathcal{D}} \left[\mathcal{L}_{\text{local}} + \lambda \mathcal{L}^{(K)}_{\text{con}} \right]. \tag{5}$$

Here S is a positive visual-channel scale estimated on the training split, sg denotes stop-gradient, and $\lambda = 1$ in all main experiments. Training windows satisfy $H \geq Kq$. The initial context, goal, and observed targets are detached from the policy optimization graph. Local supervision applies only to the initial factual decision; consequence supervision applies to all K boundaries.

Frozen dynamics, trainable decisions. Only θ is optimized. We retain gradients through F_ψ with respect to its action and context inputs while keeping (ϕ, ψ) fixed. To see the feedback contribution, write one context transition as $c_{j+1} = T_\psi(c_j, B_\theta(c_j))$, suppressing goal and horizon. The chain rule gives

$$D_{j+1} = (T_c + T_A B_c) D_j + T_A \, \partial_\theta B_\theta, \qquad D_j = \frac{\partial c_j}{\partial \theta}, \quad D_0 = 0. \tag{6}$$

Here T_c, T_A, and B_c are input Jacobians, and $\partial_\theta B_\theta$ holds the context fixed; all are evaluated at block j. The term $T_A B_c D_j$ captures feedback through later decisions. Detaching the actor's predicted-context input removes this path, while gradients through the predictor's own context remain.

Why match observations? Action regression measures disagreement in action coordinates. Consequence matching instead measures how those decisions affect the model's predicted future. Its gradients can therefore distinguish action errors by their predicted visual effects. Observed targets also prevent the supervision target from being defined entirely by the predictor's own rollout. They do not eliminate model error: an inaccurate predictor can still produce misleading gradients. The local anchor and short rollout constrain this trade-off rather than guarantee improvement.

3.2 POLICY INSTANCE AND LOCAL SUPERVISION

We instantiate the block generator with a waypoint-conditioned actor. Given state z, goal g, history h, and horizon H, it predicts a local successor and then an action output spanning b environment steps:

$$w_\theta = z + \alpha(H, b)(g - z) + r_\theta(z, g, g - z, H), \tag{7}$$

$$\hat{a}_\theta = d_\theta(z, g, w_\theta, w_\theta - z, h, H). \tag{8}$$

Here $\alpha(H, b) = b / \text{clip}(H, 1, H_{\max})$ is a horizon-scaled linear prior, which extrapolates when $H < b$; the residual corrects it in encoder coordinates. The decoder also receives state and goal directly, so the waypoint is not a hard information bottleneck. Local supervision uses the recorded action

output $a_0^{\mathcal{D}} = (a_t, \ldots, a_{t+b-1})$ and successor $z_b^{\mathcal{D}}$:

$$\mathcal{L}_{\text{local}} = \underbrace{\text{mean}\left[\left(\frac{\hat{a}_0 - a_0^{\mathcal{D}}}{s_a}\right)^2\right]}_{\mathcal{L}_a} + \underbrace{\text{mean}\left[\left(\frac{w_0 - z_b^{\mathcal{D}}}{s_w}\right)^2\right]}_{\mathcal{L}_w}. \tag{9}$$

Positive coordinate scales s_a, s_w are fixed after Stage 1. We first optimize this local objective, then retain it during consequence training. The waypoint implements the policy and its temporal adapter; the training objective, rather than waypoint prediction itself, is the proposed contribution.

3.3 Temporal alignment and direct execution

For PointMaze and Wall, the actor directly outputs a five-action block, so $b = q = 5$. The actor uses pooled visual features, while the JEPA retains its full patch and proprioceptive state. The loss supervises predicted visual patches.

For one-step action interfaces, $b = 1$ while $q = 5$. We form the block by propagating the actor's local waypoints through its short context:

$$(a_{j,r}, w_{j,r}) = \pi_\theta(\tilde{c}_{j,r}, g, H - jq - r), \qquad \tilde{c}_{j,r+1} = U_w(\tilde{c}_{j,r}, w_{j,r}), \tag{10}$$

for $r = 0, \ldots, 4$. The resulting five distinct actions enter the unchanged five-step JEPA, whose prediction supplies the next block boundary. This adapter uses waypoint predictions between boundaries; it does not assume access to a one-step JEPA.

The complete procedure consists of supervised actor pretraining followed by optimization of Equation (5), with the encoder and predictor frozen throughout. At deployment, the policy uses the encoded observation, goal, and remaining budget to predict an action or block; the environment supplies the next observation. Intermediate targets and predictive rollouts are unnecessary. Algorithm 1 appears in Appendix A.

4 Experiments

We compare complete controllers with prior work, assess coverage across 11 settings, isolate training effects with same-label controls, and evaluate their repeatability across three actor training seeds.

4.1 Experimental setup

We evaluate 11 visual control settings spanning navigation, manipulation, and puzzles, including OGBench (Park et al., 2025). Table 1 compares four tasks, with Cube denoting Cube-single; the broader survey also includes Cube-double. Reacher actors use training goal offsets up to 100 and are evaluated at $H = 25, B = 50$. Our dataset-initialized tasks use recorded-future image goals rather than the standard OGBench task-ID protocol; PointMaze and Wall use simulator-sampled goals. Appendix B specifies budgets and observation processing.

The survey and controls use 100 shared development queries per setting, grouped into five sets of 20 episodes. Their standard deviations concern reset groups, not independent training runs. Survey continuations select among epochs 1–10; the four-setting controls also admit Stage 1. Selection uses offline losses, never rollout success. Appendix C.4 reports three complete actor training repetitions in four settings; Appendices B and A specify protocols and training budgets.

Baselines and controls. Table 1 compares complete latent world-model controllers, distinguishing published results from our evaluation. In our matched experiments, GC-IDM and JACT share frozen representations, queries, and action intervals. JACT (Stage 1) and alternative training interfaces instead share the same actor and data to isolate training effects.

Performance of the complete controller. JACT achieves 97.25% average success, the highest among entries in Table 1 with complete four-task coverage, compared with 96.86% for INTACT Guarded A and 96.00% for our reproduced GC-IDM. This performance uses a direct encoder–actor policy, requiring neither predictive rollouts nor candidate search at deployment. The cross-paper

Table 1: **Comparison with prior latent world-model controllers.** Success rates (%) from the cited papers and our evaluation, with SD where reported. Averages require all four tasks; — denotes an unreported result. Cube denotes single-cube manipulation; SC denotes self-consistency scoring. Evaluation protocols and uncertainty definitions are detailed in Appendix C.1.

Method	Execution	TwoRoom	Reacher	PushT	Cube	Average
Reported results						
Latent planning						
DINO-WM	Search	100.0	79.0	74.0	86.0	84.75
LeWM	Search	87.0	86.0	96.0	74.0	85.75
Fast-LeWM	Search	98.0	88.0	96.0	80.0	90.50
Fast-LeWM + SC	Search	98.0	90.0	98.0	82.0	92.00
Qantara	Search	100.0±0.0	80.9±1.8	90.1±1.1	93.7±0.7	91.18
C-JEPA	Search	—	—	88.67	—	—
Subgoal guidance						
SAGE	Search	—	—	94.0±3.5	98.7±1.2	—
Direct policies						
GC-IDM (reported)	Direct	100.0±0.0	99.7±0.3	84.2±2.8	98.7±0.6	95.65
INTACT Direct	Direct	97.89±1.26	97.67±0.00	85.78±1.54	100.00±0.00	95.34
Policy-guided search						
INTACT Guarded A	Search	98.00±1.15	97.44±0.77	92.22±0.69	99.78±0.19	96.86
PRISM	Search	—	—	89.0±4.0	79.0±6.0	—
Predictive-objective learning						
RC-aux	Search	98.0±1.4	87.2±6.4	90.8±3.3	76.0±7.5	88.00
Our evaluation: five reset groups of 20 matched episodes						
GC-IDM (reproduced)	Direct	100.0±0.0	100.0±0.0	87.0±7.6	97.0±2.7	96.00
JACT (full)	Direct	100.0±0.0	100.0±0.0	90.0±3.5	99.0±2.2	97.25

Table 2: **Visual goal-conditioned control across 11 settings.** Success (%) ± SD over five reset groups of 20 episodes, using one trained model per setting. JACT selects among epochs 1–10. JACT (Stage 1) is our actor before consequence training, not an external baseline. Reacher uses the $H = 25, B = 50$ evaluation of actors trained with goal offsets up to 100. Cube-double uses the 50-step configuration (Appendix B).

Setting	GC-IDM	JACT (Stage 1)	**JACT (full)**
Scene	73±13.5	79±6.5	82±11.0
Cube-single	97±2.7	96±4.2	99±2.2
PointMaze	18±7.6	18±11.0	30±7.1
Wall	37±11.0	35±11.7	32±11.5
PushT	87±7.6	91±4.2	90±3.5
TwoRoom	100±0.0	100±0.0	100±0.0
Reacher	100±0.0	100±0.0	100±0.0
Puzzle-3×3	85±6.1	92±7.6	93±2.7
Puzzle-4×4	63±12.0	77±8.4	87±4.5
Cube-double	89±6.5	90±5.0	89±6.5
Scene-noisy	74±11.4	70±10.6	72±10.4
Macro average	74.82	77.09	**79.45**

differences are numerical comparisons under the source protocols, not paired significance claims (Appendix C.1). Subsequent controls examine the contribution of consequence training to this complete controller.

4.2 Exploratory Survey Across 11 Settings

Table 2 reports 11 settings using $K = 3, \lambda = 1$ and selection among second-stage epochs 1–10. JACT reaches 79.45% average success versus 77.09% for Stage 1 and 74.82% for GC-IDM, exceeding GC-IDM in six settings, tying in three, and trailing in two. Reacher is saturated at 100%. The comparison with GC-IDM includes architecture and pretraining differences; the Stage 1 com-

parison isolates continuation of our actor. The following controls also permit Stage 1 selection, giving 35% on Wall rather than the survey's 32%.

PointMaze improves from 18% at Stage 1 to 30%; Puzzle-4×4 improves from 77% to 87%. Wall instead drops from 35% to 32%, and Cube-double drops from 90% to 89%. These contrasts motivate a closer question: which gains survive stronger supervision and alternative ways to use the same future labels?

4.3 Same future labels, different training interfaces

We compare four settings spanning gains, regressions, and nearly unchanged success (Table 3). All JACT variants share Stage 1, training windows, and 100 queries per setting; continuations may retain Stage 1 and select by offline loss. JACT achieves 60.25% average success, versus 55.00% for Stage 1 and 56.00% for independent consistency. The controls distinguish future-feature prediction, independent action consequences, and recursive policy consequences (Figure 6, Appendix C.3).

Table 3: **Supervision and conditioning controls.** Success (%) ± reset-group SD on 100 queries per setting; Average equally weights the four settings. All continuations may retain Stage 1; stronger supervision and auxiliary prediction additionally search three learning rates. Wall selects Stage 1 for JACT. Appendix C.3 gives objectives and selection budgets.

Training variant	PointMaze	Wall	Puzzle-4×4	Cube-double	Average
GC-IDM	18±7.6	37±11.0	63±12.0	89±6.5	51.75
JACT (Stage 1)	18±11.0	35±11.7	77±8.4	90±5.0	55.00
Stronger supervised continuation	15±7.9	31±13.4	77±8.4	92±7.6	53.75
Independent-boundary consistency	18±11.0	35±11.7	82±7.6	89±6.5	56.00
Future-feature auxiliary supervision	12±2.7	3±4.5	75±9.4	91±8.2	45.25
JACT (full)	30±7.1	35±11.7	87±4.5	89±6.5	**60.25**

Does stronger local supervision recover the gain? Stronger supervised continuation searches three learning rates with denser validation and three times the total update budget. JACT exceeds this control by 15 percentage points on PointMaze and ten on Puzzle-4×4; Cube-double instead favors supervision by three points. On Wall, JACT retains Stage 1, so its four-point advantage over continued supervision reflects checkpoint retention, not improvement from consequence updates.

Do future-feature targets suffice? A training-only head predicts the same future targets from policy features, without passing through action outputs or dynamics. It reduces future-feature error in every setting but underperforms JACT in three of four settings, averaging 45.25%. The result favors supervising action consequences over this auxiliary-feature interface, rather than merely adding future labels. This unit-weight control is not a full FLARE reproduction (Zheng et al., 2025); auxiliary weighting and selection trade-offs remain possible explanations (Appendix C.3).

Does recursive conditioning add value beyond independent consistency? Independent-boundary consistency supplies recorded contexts to both actor and predictor at each boundary, removing inter-block coupling without mismatching their inputs. JACT improves on it by 12 points on PointMaze and five on Puzzle-4×4; Wall and Cube-double have identical outcomes. Both use the same learning rate and epoch-0–10 selection, without a separate learning-rate search. Three complete actor training repetitions retain mean gains of 5.7 and 5.3 points on PointMaze and Puzzle-4×4 (Appendix C.4). These controls support recursive supervision beyond independent prediction where the gains are strongest.

4.4 Temporal supervision and feedback gradients

Figure 2 tests rollout depth, intermediate supervision, and gradients through predicted-context feedback. All variants use the same eligible windows and allow Stage 1. On PointMaze, JACT reaches 30%, versus 18% with one block, 17% with final-boundary supervision, and 25% with stopped feedback gradients. Puzzle-4×4 also favors the full method, with smaller gaps; Wall and Cube-double

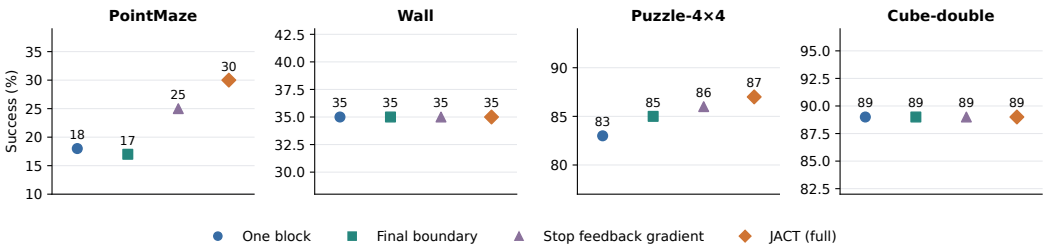

Figure 2: **Learning along the predicted trajectory.** Success on 100 shared queries per setting, with Stage-1-eligible selection as in Table 3. One block shortens the rollout; final boundary retains three blocks but supervises only the last; stopped feedback detaches predicted inputs to later actor calls. JACT supervises all three boundaries with full feedback. Axes use task-specific ranges; points show single-training-run success, not uncertainty estimates.

show no separation. Intermediate supervision and differentiable feedback contribute in the settings with the strongest gains.

Stopping feedback gradients lowers the four-setting mean by 1.50 points while preserving the forward computation at fixed weights. Predictor-history and within-block waypoint gradients remain active (Appendix D.13). Under the survey's selection rule, observed targets outperform model-generated targets by two points on PointMaze and Puzzle-4×4, while tying on PushT and Cube-double (Table 6). These controls examine the computation and its supervision separately.

4.5 Direct control and the cost of online planning

Figure 3 compares direct policies with three cross-entropy-method (CEM) budgets on 100 matched queries per setting, using Table 2's checkpoints. JACT exceeds all tested budgets on PushT, TwoRoom, Puzzle-4×4, and Cube-double. At 128×5, CEM achieves 19/81/25/59%, versus 90/100/87/89% for JACT. On PointMaze/Wall, CEM reaches 99/65% versus 30/32%, but requires 3,410/3,405 ms per decision versus 4.12/4.19 ms for JACT; each call executes five actions. Direct control offers a favorable performance–cost combination on four settings, while search improves navigation success at substantially higher cost. Appendix D.6 specifies the tested budgets and timing scopes.

4.6 Actor data and goal horizon

Learning from fewer actor-training episodes. With JEPA fixed, we retrain actors on nested data subsets (Figure 4). On Puzzle-4×4, JACT reaches 81% with 25% of actor data, exceeding same-subset Stage 1 (73%) and full-data Stage 1 (77%). It improves over equal-update supervised continuation by 6/10/8 points at 10/25/50% data. Gains over Stage 1 are smaller and mixed on Cube-double and Scene-noisy. This measures actor-stage data efficiency, not reduced world-model pretraining. Appendix D.14 specifies subsets; Appendices D.11–D.12 report training costs and time-matched controls.

One policy across goal horizons. Figure 5 varies the recorded-future goal offset H, distinct from training rollout depth K. Each method uses one fixed actor per task, trained with goal offsets up to 100; no actor is retuned across H. We separate a growing execution budget $B = 2H$ from fixed $B = 50$. On Cube-single, JACT matches or exceeds both controls in every cell; at $H = 100, B = 50$, it achieves 96% versus 94% for Stage 1 and 81% for GC-IDM. PushT has mixed gains, while Reacher and TwoRoom are near or at ceiling. These are within-range horizon tests, not extrapolation; Appendix D.10 details the shared starts and horizon-specific training recipe.

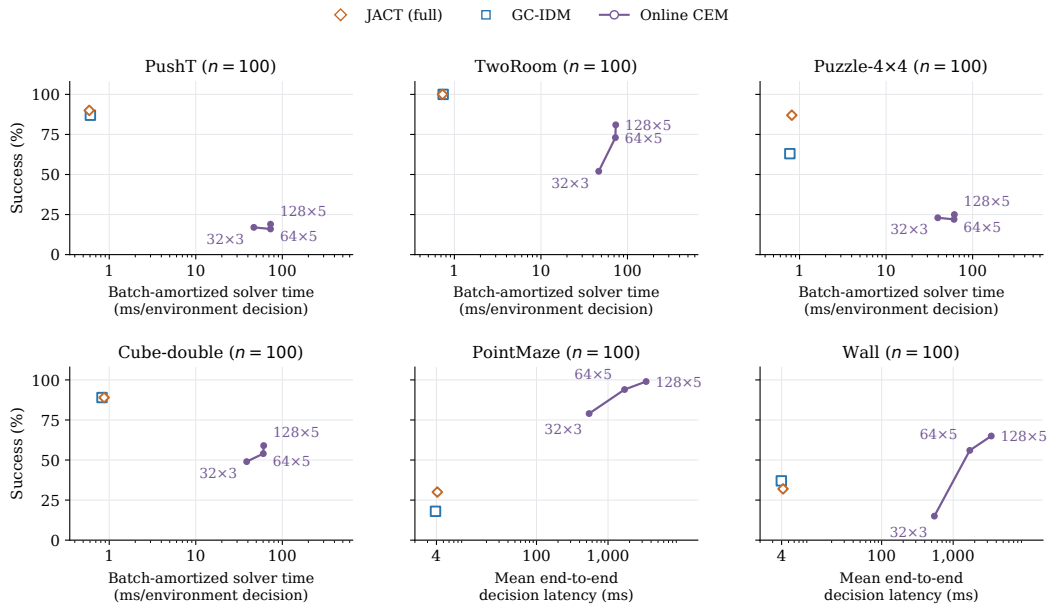

Figure 3: **Direct execution versus online search.** Success and computation on 100 matched queries per setting. CEM labels give candidates × iterations; unconnected markers denote the direct policies JACT and GC-IDM. Logarithmic time axes show batch-amortized solver time for the four LeWM settings and batch-one end-to-end decision latency for PointMaze/Wall. Compare costs within panels, not across timing scopes. JACT is competitive without deployment-time prediction; navigation illustrates when additional search improves success.

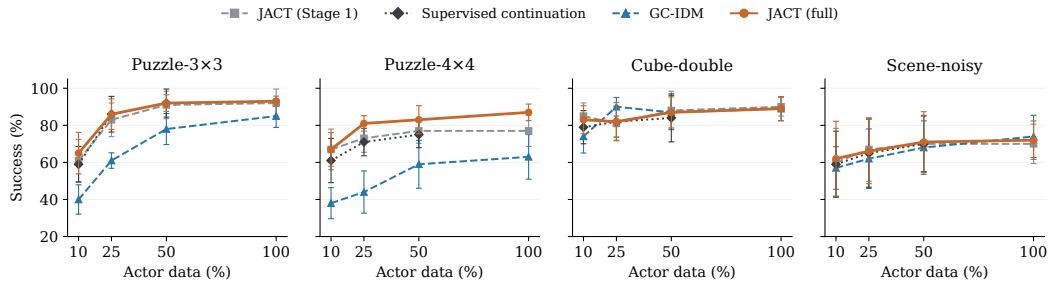

Figure 4: **Learning direct control with less actor data.** Frozen JEPA is shared; actors are trained on each subset. Points use 100 original queries and one training seed; error bars are reset-group SD. Dotted curves match continuation updates at 10/25/50%. The 100% points reuse the survey.

5 Related Work

JEPA representations and control. I-JEPA predicts target embeddings (Assran et al., 2023); DINO-WM, V-JEPA 2, and LeWM extend visual prediction to action-conditioned control (Zhou et al., 2025; Assran et al., 2025; Maes et al., 2026). Physical alignment and temporal-progress objectives improve representations for planning (Liu et al., 2026; Bai & Xiong, 2026). RC-aux adds multihorizon consistency and reachability supervision to the world model (Li et al., 2026). JACT instead freezes that model and supervises the consequences of a trainable actor.

Subgoals and amortized action generation. GC-IDM predicts actions from frozen LeWM embeddings (Nguyen et al., 2026). Predictive inverse dynamics, INTACT, and latent diffusion planning connect future representations to actions (Schäfer et al., 2026; Sun et al., 2026; Xie et al., 2025). INTACT's Direct algorithm still alternates action and forward prediction; JACT removes deployment

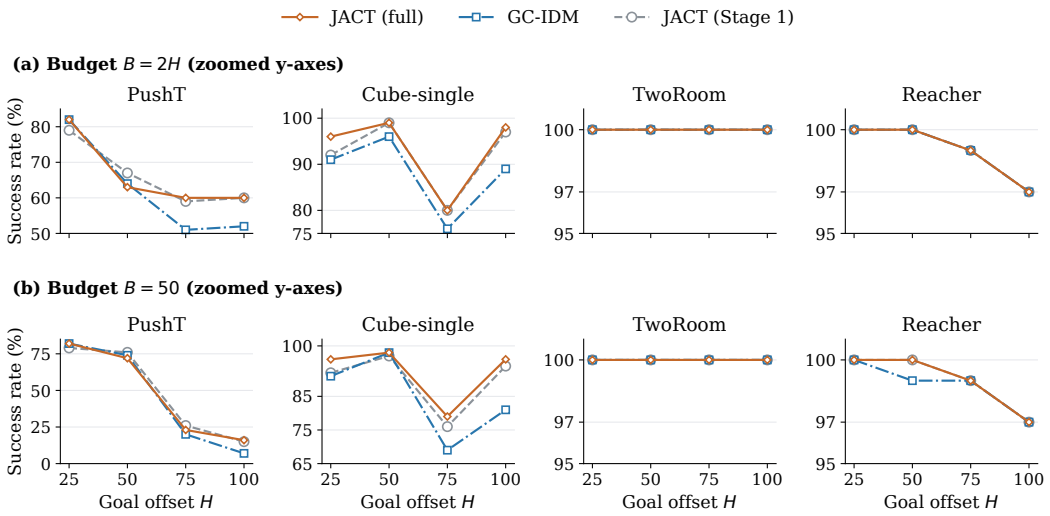

Figure 5: **One controller, different goal horizons.** Success at offsets $H = 25, 50, 75, 100$ with fixed actors and shared starts within each task. Top: $B = 2H$; bottom: $B = 50$. Both rows use task-specific y-axis ranges, including nonzero origins; all data points are retained. Temporal offsets need not imply geometric distance. TwoRoom curves coincide at 100%.

rollout. SAGE and FF-JEPA provide subgoal guidance (Cheng et al., 2026; Masip et al., 2026); hierarchical models address temporal abstraction and level compatibility (Zhang et al., 2026; Caselli et al., 2026). PRISM supplies action priors for online imagination sampling (Wang et al., 2026). Our use of dynamics is confined to offline actor training.

Future representations as policy supervision. ACT-JEPA jointly predicts actions and latent observations (Vujinović & Kovačević, 2025); FLARE aligns policy-internal futures with observed embeddings (Zheng et al., 2025). JACT instead trains actions through a frozen predictor and feeds their predicted consequences into later decisions.

Differentiable dynamics and imitation. Zero-Shot Visual Imitation jointly trains policy, representation, and dynamics through forward consistency at observed states (Pathak et al., 2018). JACT freezes dynamics and couples decisions through predicted states. Differentiating through dynamics is established in stochastic value gradients, Dreamer, and UPN (Heess et al., 2015; Hafner et al., 2020; Srinivas et al., 2018); World4RL refines policies through model-based reinforcement learning (Jiang et al., 2025). Our objective instead uses observed offline futures without reward or value fitting. Unlike interactive data aggregation (Ross et al., 2011), predicted contexts do not guarantee coverage of real policy-induced states.

6 Conclusion

JACT turns frozen JEPA dynamics into supervision for direct visual control. Its encoder–actor controller achieves the highest average among complete four-task entries in our comparison, without online prediction or search. Controlled studies and repeated actor training identify benefits on Puzzle-4×4 and PointMaze, with task-dependent limits. Predictive dynamics can shape sequential decisions during training without remaining in the deployed controller.

Limitations. The method depends on offline trajectory coverage and the accuracy of frozen dynamics; observation alignment can compensate model error rather than improve real behavior. The multirate adapter is approximate. The 11-setting survey uses one actor training realization; the three-seed evaluation covers four settings with fixed world models and training data. The results establish neither a waypoint-specific effect nor superiority over alternative predictive backbones.

REPRODUCIBILITY STATEMENT

The actor and training objective are specified in Equations (2)–(10). Appendix A gives the training algorithm and optimizer settings; Appendix B details timing and evaluation protocols; Appendix C reports additional controls, and Appendix C.4 summarizes training-seed variability and evaluation details. The method uses offline data during training and only the encoder and actor during deployment.

ETHICS STATEMENT

The experiments are conducted in simulation. The method provides no safety certificate and can fail under distribution shift. Physical deployment would require separate validation and safeguards.

AI USE STATEMENT

Generative AI assisted research discussions, experimental design, implementation and debugging, analysis of experimental records, literature discovery, mathematical exposition, manuscript drafting, and preparation of figures and tables. Reported numerical results come from experiment records, not generated data. The authors are responsible for verifying the scientific claims, references, and disclosures before submission.

REFERENCES

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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023. URL https://arxiv.org/abs/2301.08243.

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, et al. V-JEPA 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025. URL https://arxiv.org/abs/2506.09985.

Jiaxin Bai and Jiaxuan Xiong. Temporal-distance JEPA: Plan-aware representation learning for latent world model predictive control. *arXiv preprint arXiv:2607.25337*, 2026. URL https://arxiv.org/abs/2607.25337.

Niccolò Caselli, Francesco Massafra, Samuele Punzo, Salvatore Lo Sardo, Ippokratis Pantelidis, and Sathya Kamesh Bhethanabhotla. Mind the gap: Promises and pitfalls of hierarchical planning in LeWorldModel. *arXiv preprint arXiv:2607.12547*, 2026. URL https://arxiv.org/abs/2607.12547.

Letian Cheng, Qi Zhang, and Yisen Wang. SAGE: Subgoal-conditioned action generation for latent world model planning. *arXiv preprint arXiv:2607.17973*, 2026. URL https://arxiv.org/abs/2607.17973.

Yuntian Gao and Xiangyu Xu. Fast LeWorldModel. *arXiv preprint arXiv:2606.26217*, 2026. URL https://arxiv.org/abs/2606.26217.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1912.01603.

Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, volume 28, 2015. URL https://proceedings.neurips.cc/paper/2015/hash/148510031349642de5ca0c544f31b2ef-Abstract.html.

Zhennan Jiang, Kai Liu, Yuxin Qin, Shuai Tian, Yupeng Zheng, Mingcai Zhou, Chao Yu, Haoran Li, and Dongbin Zhao. World4RL: Diffusion world models for policy refinement with reinforcement learning for robotic manipulation. *arXiv preprint arXiv:2509.19080*, 2025. URL https://arxiv.org/abs/2509.19080.

Wenyuan Li, Guang Li, Keisuke Maeda, Takahiro Ogawa, and Miki Haseyama. Predictive but not plannable: RC-aux for latent world models. *arXiv preprint arXiv:2605.07278*, 2026. URL https://arxiv.org/abs/2605.07278.

Muyuan Liu, Yue Huang, Zheng Liang, and Xiang Gao. Toward physically grounded JEPA world models for goal-conditioned robotic planning. *arXiv preprint arXiv:2609.03565*, 2026. doi: 10.48550/arXiv.2609.03565. URL https://arxiv.org/abs/2609.03565.

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

Sergi Masip, Jonathan Swinnen, Yutong Hu, Renaud Detry, and Tinne Tuytelaars. FF-JEPA: Long-horizon planning in world models with latent planners. *arXiv preprint arXiv:2606.09311*, 2026. URL https://arxiv.org/abs/2606.09311.

Heejeong Nam, Quentin Le Lidec, Lucas Maes, Yann LeCun, and Randall Balestriero. Causal-JEPA: Learning world models through object-level latent masking. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2602.11389.

Hoang Nguyen, Xiaohao Xu, and Xiaonan Huang. Latent geometry beyond search: Amortizing planning in world models. *arXiv preprint arXiv:2605.08732*, 2026. URL https://arxiv.org/abs/2605.08732.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. OGBench: Benchmarking offline goal-conditioned RL. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.20092.

Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A. Efros, and Trevor Darrell. Zero-shot visual imitation. In *International Conference on Learning Representations*, 2018. URL https://pathak22.github.io/zeroshot-imitation/resources/iclr18.pdf.

Ruslan Rakhimov, George Bredis, Yuriy Maksyuta, and Daniil Gavrilov. Qantara: Bridge-flow training for multi-paradigm JEPA control. *arXiv preprint arXiv:2607.04978*, 2026. URL https://arxiv.org/abs/2607.04978.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635. PMLR, 2011. URL https://proceedings.mlr.press/v15/ross11a.html.

Lukas Schäfer, Pallavi Choudhury, Abdelhak Lemkhenter, Chris Lovett, Somjit Nath, Luis França, Matheus Ribeiro Furtado de Mendonça, Alex Lamb, Riashat Islam, Siddhartha Sen, John Langford, Katja Hofmann, and Sergio Valcarcel Macua. When does predictive inverse dynamics outperform behavior cloning? In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2601.21718.

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 4732–4741, 2018. URL https://proceedings.mlr.press/v80/srinivas18b.html.

Junhan Sun, Hao Zhao, and Guofeng Zhang. INTACT: Isomorphic intent-to-action learning for search-free world models. *arXiv preprint arXiv:2607.26056*, 2026. URL https://arxiv.org/abs/2607.26056.

Aleksandar Vujinović and Aleksandar Kovačević. ACT-JEPA: Novel joint-embedding predictive architecture for efficient policy representation learning. *arXiv preprint arXiv:2501.14622*, 2025. URL https://arxiv.org/abs/2501.14622. Version 5, June 2026.

Yuhai Wang, Jiawei Xia, Rongxuan Zhou, Xiao Hu, Yongliang Shi, Jing Du, and Yang Ye. PRISM: PRior-guided imagination sampling in world models. *arXiv preprint arXiv:2606.07974*, 2026. URL https://arxiv.org/abs/2606.07974.

Amber Xie, Oleh Rybkin, Dorsa Sadigh, and Chelsea Finn. Latent diffusion planning for imitation learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 68710–68724, 2025. URL https://proceedings.mlr.press/v267/xie25h.html.

Wancong Zhang, Basile Terver, Artem Zholus, Soham Chitnis, Harsh Sutaria, Mido Assran, Randall Balestriero, Amir Bar, Adrien Bardes, Yann LeCun, and Nicolas Ballas. Hierarchical planning with latent world models. *arXiv preprint arXiv:2604.03208*, 2026. URL https://arxiv.org/abs/2604.03208.

Ruijie Zheng, Jing Wang, Scott Reed, Johan Bjorck, Yu Fang, Fengyuan Hu, Joel Jang, Kaushil Kundalia, Zongyu Lin, Loic Magne, Avnish Narayan, You Liang Tan, Guanzhi Wang, Qi Wang, Jiannan Xiang, Yinzhen Xu, Seonghyeon Ye, Jan Kautz, Furong Huang, Yuke Zhu, and Linxi Fan. FLARE: Robot learning with implicit world modeling. *arXiv preprint arXiv:2505.15659*, 2025. doi: 10.48550/arXiv.2505.15659. URL https://arxiv.org/abs/2505.15659.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 79115–79135, 2025. URL https://proceedings.mlr.press/v267/zhou25t.html.

A TRAINING ALGORITHM AND IMPLEMENTATION

Algorithm 1 JACT on one offline minibatch

Require: Frozen encoder E_ϕ and predictor F_ψ; actor π_θ initialized with $\mathcal{L}_{\text{local}}$; offline minibatch

Load detached initial context c_0, goal g, horizon H, and observed future targets
for $j = 0, \ldots, K-1$ **do**
    Generate a q-step action block $A_j = B_\theta(c_j, g, H - jq)$
    Cache the first actor output when $j = 0$ for local supervision
    Predict $\hat{x}_{j+1} = F_\psi(c_j, A_j)$, retaining input gradients
    Update context $c_{j+1} = U(c_j, A_j, \hat{x}_{j+1})$ without detaching predictions
end for
Compute $\mathcal{L}_{\text{local}}$ from the cached initial output (Equation (9))
Compute $\mathcal{L}_{\text{con}}^{(K)}$ against detached observations (Equation (4))
Update θ on $\mathcal{L}_{\text{local}} + \lambda \mathcal{L}_{\text{con}}^{(K)}$
Keep (ϕ, ψ) unchanged

Actor and world-model interfaces. The actor has separate waypoint and action multilayer perceptrons. Its action decoder also receives state, goal, horizon, and history context. Within each setting, the deployed actor architecture, feature normalization, and horizon clipping are fixed across both training stages and all objective ablations. Empirical action means enter the output map, while the action loss uses coordinate-wise action scales. The maximum horizon for each setting is specified in Table 5.

The predictive context contains the state and action history required by the world model. For PointMaze and Wall, P averages visual patches over views and spatial positions and normalizes the resulting vector; Q retains full visual patches. Proprioception remains in the dynamics state even though the consequence loss supervises visual channels. World-model action normalization and actor-loss normalization are distinct. These rollouts recompute short action prefixes from the initial context, preserving the world model's history-dependent interface and gradients through earlier actions.

Optimization. Unless explicitly varied in a control, the second stage starts from the Stage 1 actor and uses seed 42, ten epochs of AdamW with learning rate 10^{-4}, weight decay 0.01, a cosine schedule to zero, and gradient-norm clipping at 1. Batch size is 128 for one-step action interfaces and 32 for PointMaze/Wall. Action, waypoint, and consequence losses have unit weight. Encoder and predictor weights remain fixed in evaluation mode, but their input derivatives remain enabled in the actor-generated branch. The initial actor output is cached so that local supervision and the first rollout action share the same stochastic forward pass.

Training windows. Each second-stage window stays within a single episode and training or offline validation partition, has continuous raw-step indices, and admits three complete prediction blocks with $H \geq 15$. One-block controls use the same windows. During training, H is the recorded goal offset; at deployment it is the remaining execution budget. These need not coincide (e.g., a 25-step image goal with a 50-step budget). The visual target scale is estimated from training data; actor normalization statistics are fixed after Stage 1. Each second-stage variant selects its checkpoint using its own offline validation objective, rather than evaluation success. Equal data and update budgets do not imply identical selection criteria or computational cost.

Stage 1 minimizes standardized action and successor losses and selects among 50 epochs by their joint validation objective. Its cosine schedule ends at 10^{-5}; the stage-2 schedule ends at zero. The actor has width 512, depth three, and dropout 0.1. Navigation Stage 1 uses a window-level split and normalization statistics over its training and validation windows. Puzzle/Cube Stage 1 uses a 900/100 episode split and fit-only normalization. Stage 2 uses episode-disjoint fit and validation windows in all four settings. These actor-data guarantees do not establish disjointness from frozen-world-model pretraining. Stage-2 normalization is inherited from Stage 1; consequence-channel scales use stage-2 fit data only.

Table 4: **Two-stage configuration for the four core settings.** Stage 1 uses the full supervised pretraining recipe, not the smaller stage-2 window set. All optimizers are AdamW. Stage-2 values describe the fixed JACT recipe; Table 3 specifies the broader tuning budgets of the stronger controls.

	PointMaze	Wall	Puzzle-4×4	Cube-double
Stage 1 fit examples	171,000	79,488	900,000	900,000
Stage 1 validation examples	19,000	8,832	100,000	100,000
Stage 1 epochs / batch	50 / 1024	50 / 1024	50 / 1024	50 / 1024
Stage 1 learning rate / decay	$10^{-3}/10^{-4}$	$10^{-3}/10^{-4}$	$10^{-3}/10^{-4}$	$10^{-3}/10^{-4}$
Stage 1 maximum goal offset	50	50	25	25
Stage 2 fit / validation windows	4096 / 512	4096 / 512	32768 / 4096	32768 / 4096
Stage 2 epochs / batch	10 / 32	10 / 32	10 / 128	10 / 128
Stage 2 learning rate / decay	$10^{-4}/10^{-2}$	$10^{-4}/10^{-2}$	$10^{-4}/10^{-2}$	$10^{-4}/10^{-2}$
Stage 2 optimizer updates	1280	1280	2560	2560

Data partitions and normalization. Both Puzzle/Cube actor stages split each official training file into 900 fit and 100 checkpoint-validation trajectories. Actor statistics use fit rows, and consequence scales use second-stage fit windows. Frozen JEPA action statistics use the whole training file; JEPA pretraining uses an internal 90/10 window split and epoch 50, without the official validation file. Navigation Stage 1 retains its original window-level split and normalization over all eligible windows; its second-stage split is episode-disjoint. Complete pretraining source identities of the inherited navigation models are unavailable.

Controlled variants. The Stage-1-only ablation omits the second stage. The ablation without consequence loss retains that stage but minimizes only $\mathcal{L}_{\text{local}}$. The model-generated target ablation substitutes a detached rollout of recorded actions for the observed targets:

$$\mathcal{L}^{(K)}_{\text{model}} = \text{mean}\left[\left(\frac{R_\psi(c_0, A_\theta) - \text{sg}(R_\psi(c_0, A_\mathcal{D}))}{S}\right)^2\right], \tag{11}$$

where R_ψ collects all visual rollout boundaries. The recorded-state feedback control replaces only the actor's input at each model boundary with its recorded counterpart. The world-model trajectory remains action-driven; within-block waypoint propagation for one-step action interfaces is unchanged. This intervention can mismatch actor and predictor contexts and should not be interpreted as independent factual-state forward consistency. One-block training retains the same normalization and local losses but uses only the first predicted boundary.

B Evaluation protocol

The survey and single-seed controls share their start–goal queries. Survey continuations select among epochs 1–10, whereas core controls also admit Stage 1; this changes the Wall checkpoint from 32% to 35%. Supplementary analyses retain the survey checkpoints unless otherwise stated. The three-training-seed evaluation uses a separate set of 100 shared queries per task, described in Appendix C.4. Scene, Cube-single, PointMaze, Wall, PushT, TwoRoom, and Puzzle-3×3 use reset seeds 9200801, 9200811, 9200821, 9200831, and 9200841. Reacher uses 9201301, 9201311, 9201321, 9201331, and 9201341. Puzzle-4×4, Cube-double, and Scene-noisy use 42, 52, 62, 72, and 82. Each seed defines a group of 20 episodes; these are reset groups, not training repetitions. Dataset-initialized tasks use matching start rows and goals. PointMaze and Wall use matching randomly sampled initial/goal states and encoded-input identities. All methods have the same execution budget and action interval within each setting.

Puzzle-4×4, Cube-double, and Scene-noisy use official OGBench datasets with the goal offsets below. Puzzle-3×3 and the three official-data settings use 224-pixel FP32 model inputs produced from 64-pixel camera observations, identically for every compared method. For the official-data settings, physical reset states and policy inputs match exactly; a predefined tolerance applies only to an auxiliary live RGB render, not to policy inputs.

Table 5: Evaluation time scales for the settings in Table 2, in environment steps (wrapper steps for Reacher). Goal offset, episode budget, and actor horizon clipping are separate quantities; "—" indicates a simulator-sampled goal without a trajectory offset.

Setting	Episode budget	Goal offset	Steps / decision	Maximum H
Scene	50	25	1	50
Cube-single	50	25	1	50
PointMaze	30	—	5	50
Wall	30	—	5	50
PushT	50	25	1	50
TwoRoom	50	25	1	50
Reacher	50	25	1	100
Puzzle-3×3	50	25	1	25
Puzzle-4×4	50	25	1	25
Cube-double	50	25	1	25
Scene-noisy	50	25	1	25

Episode-disjoint fit/validation partitions apply to the second-stage windows, not every Stage 1 or pretraining stage. The survey queries were used during method development, and their standard deviations concern resets of a fixed trained controller.

C Additional experimental results

C.1 Literature comparison settings

Fast-LeWM and its self-consistency (SC) variant use Table 1 of Gao & Xu (2026), with ten training epochs and a five-block planning horizon; SC changes candidate scoring with weight one. Qantara uses the latent-planning row of Table 1 of Rakhimov et al. (2026), with three training seeds and four denoising steps per transition in the authors' reproduction pipeline. C-JEPA uses the PushT result in Table 6 of Nam et al. (2026), with object-centric features, one masked object, and three seeds of 50 trajectories. Its representation and evaluation setup differ from the LeWM-suite rows. Fast-LeWM and C-JEPA do not tabulate numeric success-rate SD. All averages are recomputed from displayed task means. We omit INTACT's separate one-epoch reproductions of GC-IDM and PRISM, which answer an early-training comparison rather than report those methods' full-training results. The reported-results block of Table 1 compiles measurements from the cited papers. DINO-WM and LeWM use the means in Figure 6 of Maes et al. (2026); the DINO-WM row is the evaluation conducted by those authors, not the original DINO-WM task suite. We do not infer numeric SD from plotted error bars. SAGE uses its $H = 25$ rows (Table 1), with three 50-query manifests and execution budgets $2H$ on PushT and H on Cube. PRISM uses LeWM-based MPPI with 128 samples and 30 iterations on both tasks (Table 1), evaluated on three seeds with 50 episodes each. Missing tasks are not imputed. GC-IDM uses the $n = 200$ rows of its Table 1, not its separate $n = 50$ experiment. INTACT uses its goal-displacement Direct and Guarded A rows (Table 2): SD is across three training seeds, each averaged over three evaluation seeds of 100 episodes. Guarded A performs 128×3 candidate evaluations. RC-aux uses four task columns from its Table 1, with five fixed evaluation groups. These statistics differ from our five reset groups for one trained model; cross-study differences are not paired effect estimates. In the lower group, Cube denotes our Cube-single setting, not Cube-double.

The within-study Reacher column uses the $H = 25, B = 50$ cell of Appendix D.10: both GC-IDM and JACT achieve $100.0 \pm 0.0\%$ on the same 100 queries, grouped into five sets of 20. These actors use training goal offsets up to 100; they are the fixed actors used for the goal-horizon study, not an additional run of the shorter-range survey recipe. The four-task averages in Table 1 are arithmetic means over its displayed columns; Table 2 uses the same Reacher cell for GC-IDM, Stage 1, and JACT, and includes only the 50-step Cube-double configuration.

C.2 Training-objective ablations across all settings

Table 6 separates Stage 1 pretraining from ablations of the second training stage, on the same settings and queries as Table 2, with identical Stage 1 and full-method checkpoints. JACT (Stage 1) has a smaller training budget; the other variants share second-stage data and update counts. The two additional Reacher controls are unevaluated, so their cells and incomplete-column averages are omitted.

Table 6: **Ablations of policy training.** The same 11 settings and evaluation queries as Table 2; success (%) ± reset-group SD. JACT (Stage 1) omits the second stage; the other variants share its update budget. Without consequence loss uses local supervision only; model-generated targets replace observed futures with detached predictions under recorded actions. — denotes an unevaluated control; macro averages require all 11 settings.

Setting	JACT (Stage 1)	Without consequence loss	Model-generated targets	**JACT (full)**
Scene	79±6.5	81±8.2	79±9.6	82±11.0
Cube-single	96±4.2	94±5.5	97±2.7	99±2.2
PointMaze	18±11.0	10±3.5	28±5.7	30±7.1
Wall	35±11.7	4±4.2	30±11.7	32±11.5
PushT	91±4.2	93±4.5	90±7.9	90±3.5
TwoRoom	100±0.0	100±0.0	100±0.0	100±0.0
Reacher	100±0.0	—	—	100±0.0
Puzzle-3×3	92±7.6	93±5.7	93±2.7	93±2.7
Puzzle-4×4	77±8.4	76±8.9	85±5.0	87±4.5
Cube-double	90±5.0	88±7.6	89±6.5	89±6.5
Scene-noisy	70±10.6	71±11.9	72±10.4	72±10.4
Macro average	77.09	—	—	**79.45**

These ablations retain epoch-1–10 selection, whereas the four-setting mechanism comparisons in the main text also admit Stage 1. Recorded-state feedback replaces actor inputs only and can mismatch the actor and predictor contexts; it is not the independent factual-boundary control in Table 3.

C.3 Supervision and conditioning controls

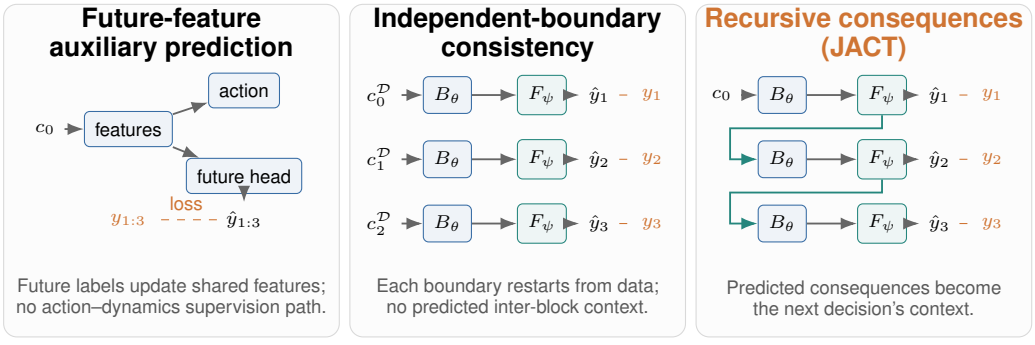

Figure 6: **Identical future labels supervise different computations.** Auxiliary prediction attaches targets to policy features, independent consistency supervises actions from factual contexts, and JACT couples successive decisions through predicted contexts. $\hat{y}_j$ is a model output; y_j is a detached observed-future target. Solid arrows show forward computation; orange dashed links compare predictions with targets in the loss. Targets never enter subsequent actor calls. The shared local anchor is omitted.

We revisit PointMaze, Wall, Puzzle-4×4, and Cube-double, retaining each setting's initialized actor, training windows, channel scales, and original 100 evaluation queries. PointMaze/Wall use 4,096 training and 512 offline validation windows, batch size 32, and 1,280 updates per ten-epoch run. Puzzle/Cube use 32,768/4,096 windows, batch size 128, and 2,560 updates. All use training seed

Table 7: **Ablations of JACT.** Single-training-seed success (%) on the original 100 development queries; all continuations select among epochs 1–10, unlike the Stage-1-eligible core controls. The highlighted row uses observed targets, predicted-state feedback, and $K = 3$. Training windows and updates match. Recorded-state feedback changes actor inputs only. Puzzle: Puzzle-4×4; Cube: Cube-double.

Training variant	PointMaze	Wall	Puzzle	Cube	Average
JACT (Stage 1)	18	35	77	90	55.00
JACT (full)	30	32	87	89	59.50
Without consequence loss	10	4	76	88	44.50
One prediction block ($K = 1$)	10	3	83	89	46.25
Recorded-state feedback	24	25	82	85	54.00
Stopped feedback gradient	25	28	86	89	57.00
Model-generated targets, $K = 3$	28	30	85	89	58.00
Model-generated targets, $K = 1$	14	5	84	89	48.00

42, AdamW with weight decay 0.01, and the original cosine schedule. Every variant admits Stage 1 and minimizes its own offline validation loss, never rollout success. Table 3 reuses evaluated policies under this selection rule; no new test episodes are added.

Stronger supervised continuation. We optimize only the original action and waypoint losses at learning rates 10^{-4}, 3×10^{-5}, and 10^{-5}. The complete offline validation set is evaluated every 16 updates. Selection minimizes local validation loss across learning rates and checkpoints, including update zero. The grid uses three times the main method's updates: 3,840 on PointMaze/Wall and 7,680 on Puzzle/Cube, not an equal-compute comparison. PointMaze and Wall select 10^{-5} at updates 288 and 336, respectively. Offline loss changes only slightly, from 1.59753 to 1.59686 and from 1.26188 to 1.25973. Both remain below Stage 1 in control success. Wall reaches 31%, four points below the Stage 1 selected by JACT; this difference cannot establish a benefit of consequence updates. Puzzle and Cube select 10^{-4} at updates 16 and 256, with local loss decreasing from 1.02508 to 1.02320 and 1.12357 to 1.10818. Puzzle matches Stage 1 at 77%, whereas Cube improves it from 90% to 92% and exceeds JACT (89%).

Independent factual boundaries. For each $j \in \{0, 1, 2\}$, both actor and frozen predictor receive their required recorded context $c_j^{\mathcal{D}}$. The actor generates $A_j = B_\theta(c_j^{\mathcal{D}}, g, H - 5j)$ and the prediction is matched to the same observed target at boundary $j + 1$. No predicted context or action prefix passes between boundaries. Navigation uses the complete recorded visual and proprioceptive state. Puzzle/Cube retain the predictor's factual state and action history, while the actor reads the latest recorded latent and generates five one-step actions through its original within-block waypoint adapter. Only the inter-block conditioning is changed. The local anchor remains at $j = 0$; the three normalized future losses are averaged with unit weight. Unlike actor-only recorded-state feedback, this does not mismatch the modules' contexts. It is a controlled forward-consistency objective, not a reproduction of the joint representation/model training in Pathak et al. (2018).

This control uses learning rate 10^{-4} and selects the minimum offline total loss among epochs 0–10, as does JACT in the four-setting comparison; ties favor the earlier epoch. It selects Stage 1 on PointMaze and Wall and epoch 1 on Puzzle and Cube, reaching 18/35/82/89% success. The comparison does not establish the best independent-boundary performance under learning-rate tuning. Navigation uses three predictor transitions per window versus six in the original prefix-replay implementation; this is not compute-matched. Numerical checks confirm identical one-boundary losses and actor gradients, no dependency on earlier boundary predictions, and unchanged frozen-model weights.

Episode-level comparison. Relative to stronger supervision, the Stage-1-eligible JACT wins/loses/ties on 17/2/81 PointMaze, 6/2/92 Wall, 10/0/90 Puzzle, and 0/3/97 Cube queries. Relative to independent boundaries, the respective counts are 15/3/82, 0/0/100, 5/0/95, and 0/0/100. Relative to independent consistency, PointMaze and Puzzle retain gains, whereas Wall and Cube-double have identical outcomes. All comparisons concern one training realization per variant and the original development queries, not independent training repetitions or a held-out final test.

Future-feature auxiliary supervision. To test whether additional future labels alone reproduce the observed gains, we attach a trainable prediction head to the shared action-decoder backbone, before the action-output layer. For PointMaze and Wall, the head receives the policy's hidden activation and detached initial visual patches, with learned spatial and future-boundary embeddings. A width-128 patchwise MLP predicts residuals relative to the initial patches at raw steps 5, 10, and 15. The initial-patch input and skip connection preserve spatial information that is lost in the actor's pooled input. The targets are exactly the three full patch tensors used by JACT, not pooled substitutes. Future observations enter only the loss. Neither actor action outputs nor the world-model predictor are inputs to this head; its gradients update the shared policy representation. This is an auxiliary-supervision control motivated by future-representation learning (Zheng et al., 2025), not a reproduction of FLARE's architecture and training procedure.

On PointMaze/Wall, the head adds 279,552 training-only parameters to the 2,898,314-parameter actor. It combines projected policy and initial-patch features with learned position embeddings, followed by layer normalization, GELU, a width-128 linear layer, GELU, and the output projection. Puzzle/Cube use their original projected latent-vector targets, not navigation patch tensors: the same decoder receives the first actor call's shared hidden activation and detached initial latent, with three boundary embeddings. It adds 132,288 parameters to each 2,010,821-parameter actor. A small random output initialization approximates persistence while retaining nonzero auxiliary gradients. We optimize the unchanged local loss plus unit-weight normalized future-feature MSE; deployment discards the head. Gradient checks verify that the auxiliary objective reaches the shared backbone but not the action-output weights, and that changing future labels or action-output weights does not change auxiliary predictions.

This control uses the same windows, Stage 1, seed, batch size, optimizer, and three-learning-rate budget as stronger supervised continuation. Offline validation occurs every 16 updates and includes update zero. Selection minimizes local plus auxiliary loss, rather than local loss alone, across 3,840 updates on PointMaze/Wall and 7,680 on Puzzle/Cube. All select learning rate 10^{-4}, at updates 1,216/1,264/2,560/2,560, respectively. Search budgets match stronger supervision, but selection objectives, parameter counts, and computation do not. No rollout success is used for selection.

Table 8: **Better future-feature prediction need not improve control.** Offline validation losses before training and at the selected auxiliary checkpoint. Local loss is the sum of normalized action and waypoint MSEs. W/L/T counts JACT wins, losses, and ties relative to auxiliary supervision on the original 100 queries.

Setting	Future-feature MSE	Local loss	W/L/T
PointMaze	$1.267 \rightarrow 1.019$	$1.598 \rightarrow 1.605$	21/3/76
Wall	$1.504 \rightarrow 1.183$	$1.262 \rightarrow 1.273$	32/0/68
Puzzle-4×4	$1.511 \rightarrow 1.082$	$1.025 \rightarrow 1.067$	12/0/88
Cube-double	$3.695 \rightarrow 3.272$	$1.124 \rightarrow 1.129$	0/2/98

The auxiliary policy achieves 12% on PointMaze and 3% on Wall (Table 3); Stage-1-eligible JACT wins/loses/ties on 21/3/76 and 32/0/68 queries, respectively. Auxiliary validation MSE decreases by 19.6% and 21.4%, but local validation loss increases (Table 8). Selected training/validation future-patch MSEs are 1.015/1.019 on PointMaze and 1.164/1.183 on Wall. Thus the head learns predictive information, yet its total-loss improvement does not translate into better control. This is consistent with an auxiliary-task trade-off or checkpoint-selection mismatch, not proof of either explanation. The fixed unit weight and particular decoder do not establish the best possible auxiliary baseline. On Wall, JACT selects Stage 1 (35%); the large auxiliary gap is not evidence that consequence updates help.

Puzzle/Cube auxiliary success is 75/91%, versus 87/89% for JACT, with JACT W/L/T counts of 12/0/88 and 0/2/98. Future-feature validation error decreases by 28.4% and 11.4%, respectively, while local loss increases in both. The relation between these offline losses and control is therefore not uniform: Cube's auxiliary policy improves over Stage 1 and JACT, despite its higher local loss. Together with its stronger-supervision result and identical independent-boundary outcomes, Cube provides no evidence that recursive consequence supervision is necessary for high success under this protocol.

C.4 Evaluation across actor training seeds

We evaluate complete actor training seeds 42, 52, and 62 on PointMaze, Wall, Puzzle-4×4, and Cube-double. Each seed uses the original full 50-epoch Stage 1 pretraining, followed by the second-stage configurations in Appendix C.3. Within each seed, the three continuation methods share that initialized actor. JACT and independent consistency use learning rate 10^{-4} and select among Stage 1 and ten epochs by their own offline validation objectives. Stronger supervised continuation retains its three-learning-rate search and denser validation. Ties favor earlier checkpoints. Data partitions and pretrained world models remain fixed; the repetitions measure actor optimization variability. No auxiliary-head or additional GC-IDM training repetitions are included in this comparison.

Table 9: **Performance across actor training seeds.** Success (%) ± sample SD over three complete actor training runs, each evaluated on the same 100 queries per setting. World models remain fixed. Cube-double uses 50 steps. All continuations allow Stage 1 in offline checkpoint selection.

Training variant	PointMaze	Wall	Puzzle-4×4	Cube-double
JACT (Stage 1)	14.7±2.5	27.7±1.5	73.7±2.1	86.0±1.0
Stronger supervised continuation	17.0±2.6	25.3±0.6	74.3±2.5	87.3±2.1
Independent-boundary consistency	16.3±5.1	27.7±1.5	79.7±3.5	87.0±1.0
JACT (full)	22.0±6.0	26.3±2.5	85.0±1.7	88.0±1.0

Repeatability of the training gains. JACT improves over Stage 1 by 7.3 points on PointMaze and 11.3 on Puzzle, and over independent consistency by 5.7 and 5.3 points, respectively. All three Puzzle runs exceed every control; PointMaze improves over Stage 1 in each run but is more variable against stronger supervised continuation and independent consistency. Cube-double's mean gains are two points over Stage 1 and one over independent consistency; Wall is 1.3 points below Stage 1. The results support repeatable benefits on Puzzle and PointMaze without implying uniform gains across tasks.

Evaluation settings. All selected actors are frozen before constructing queries. PointMaze and Wall use 100 newly seeded simulator resets and goals, checked against recorded earlier queries. Puzzle and Cube draw one nontrivial start/goal pair from each of the 100 trajectories in their existing official validation files, excluding previously recorded start and goal rows. Goals remain 25 steps ahead, with a 50-step execution budget. Puzzle requires a change in button states; Cube retains the original minimum change criterion, $\|q_{t+25}[9:] - q_t[9:]\|_2 \geq 0.04$. Sampling uses fixed random seeds and no policy outcomes. Every trained actor receives the same 100 queries within its setting. Query randomization is separate from actor training seeds.

The Puzzle/Cube evaluation trajectories are separate from actor fitting and checkpoint selection in both stages, but their trajectory pool was used during earlier development. The comparison measures actor-training variability, not generalization to a previously untouched trajectory pool.

After a redundant live-RGB assertion interrupted one 20-query Puzzle batch, we changed it to logging only and reran all 12 actors on that batch. This removes only the diagnostic exception: physical reset preparation, policy pixels, goals, random seeding, action execution, and success/termination checks are unchanged. Exact non-diagnostic context checks remain active. All 100 queries are retained; the other 80 keep the original diagnostic.

We first pool each trained actor's 100 binary outcomes, then compute the mean and sample standard deviation of the three actor-level success percentages. Differences between methods are computed before rounding. Thus Table 9 measures training-seed variation, unlike the reset-group statistics elsewhere. The three actors share queries, so their results do not constitute 300 distinct evaluation queries. PointMaze's mean advantage does not hold in every comparison: one run loses four points to stronger supervised continuation and another ties independent consistency. These repetitions complement the original-query comparisons; both training realization and evaluation queries affect absolute success.

C.5 CAPACITY-MATCHED ACTOR DIAGNOSTIC

Both actors start from common random weights, use identical normalization and data order, and train for 50 epochs. This diagnostic uses the restricted second-stage window sets below, not the full-data Stage 1 recipe in the main experiments. Its lower absolute success therefore cannot be substituted for the main-table Stage 1. The explicit arm uses a waypoint prior and successor supervision; the generic arm uses the same modules as an intermediate feature with action supervision alone. Both select checkpoints by offline action MSE. This changes intermediate semantics and supervision, not topology; neither arm uses consequence training.

Table 10: **Waypoint parameterization under local supervision.** Success (%) $\pm$ reset-group SD. Both actors are trained from scratch without consequence training; neither is the complete method.

Setting	Explicit waypoint	Generic intermediate feature
PointMaze	8 ± 4.5	17 ± 8.4
Wall	4 ± 6.5	6 ± 2.2
Puzzle-4×4	7 ± 4.5	8 ± 4.5
Cube-double	54 ± 20.4	55 ± 14.1

PointMaze/Wall have 2,898,314 parameters per arm, 4,096 training and 512 validation windows, and 6,400 updates; Puzzle-4×4/Cube-double have 2,010,821 parameters, 32,768/4,096 windows, and 12,800 updates. Selected epochs (explicit/generic) are 4/44, 48/38, 5/3, and 10/6, respectively. The generic arm has higher success throughout. This does not establish a waypoint advantage or test its interaction with consequence training.

D FURTHER ANALYSES OF CONSEQUENCE TRAINING

D.1 WHERE TO PLACE TEMPORAL SUPERVISION

We hold the three-block rollout, predicted-state feedback, local anchor, and $\lambda = 1$ fixed, and replace the average over three observed-future boundaries in Equation (4) with supervision of the final boundary only. The final-boundary loss is not divided by three, so this comparison changes the distribution of supervision rather than introducing an additional factor-of-three reduction in its weight. Both variants use ten second-stage epochs and the same eligible training windows, with checkpoint selection by their respective offline validation objectives.

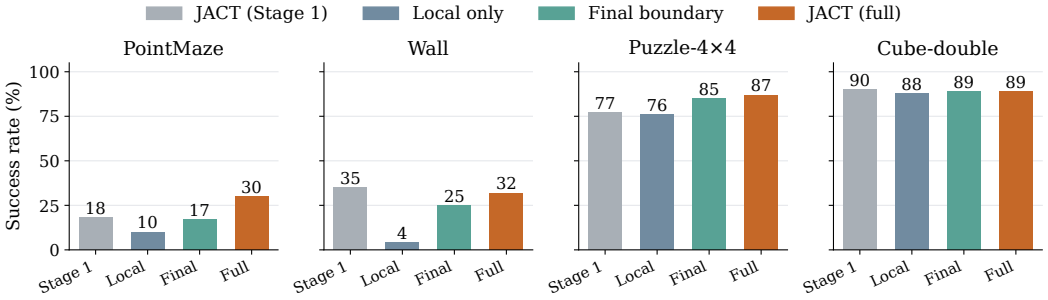

Figure 7: **Distributing supervision across predicted consequences.** Success on the original 100 development queries per setting, with one training realization and epoch-1–10 selection, as in the survey; Stage 1 is not a continuation candidate. Final-boundary supervision and complete JACT use the same three-block rollout; JACT supervises all three boundaries. Local-only continuation matches second-stage updates. Orange denotes the full method.

Evaluation uses all five original reset groups, with 100 matched episodes and one training realization per setting. The complete-method controls are identical to those in Table 7. Compared with final-boundary supervision, supervising all boundaries improves success from 17 to 30% on PointMaze, 25 to 32% on Wall, and 85 to 87% on Puzzle, while tying at 89% on Cube. The larger gains are concentrated in navigation; the two-point Puzzle difference alone provides limited evidence of a benefit.

D.2 APPLYING CONSEQUENCE TRAINING TO A WAYPOINT-FREE ACTOR

We apply the same three-block JEPA consequence objective to GC-IDM, retaining its original five-action output, horizon input, and raw-action regression loss. No waypoint head or waypoint loss is added. Starting from the same GC-IDM checkpoint, we compare ten epochs of action-supervised continuation with an otherwise identical continuation that adds the observed-consequence loss. Data, update counts, stochastic actor calls, and checkpoint-selection procedure are matched within this pair; each uses its own offline validation objective.

Table 11: **Consequence training applied to GC-IDM.** Success (%) on the original 100 matched queries per setting. The two continuation variants share their initialized GC-IDM, data and update budget; neither uses a waypoint head.

Setting	Original GC-IDM	Action-only continuation	With consequence loss
PointMaze	18	12	22
Wall	37	21	33

Consequence supervision improves on equal-update action-only continuation from 12 to 22% on PointMaze and from 21 to 33% on Wall. Relative to the original GC-IDM checkpoint, however, it gains four successes in PointMaze (18 to 22%) and loses four in Wall (37 to 33%). The effect is therefore partly protection against degradation during further optimization, not a uniform gain over the starting policy. GC-IDM and JACT retain different local losses and architectures; this experiment is not an isolated test of waypoint structure.

D.3 Offline action and consequence alignment

For each setting we select 128 fixed windows from the second-stage offline validation partition, shared by all evaluated checkpoints. No optimization or simulator interaction occurs in this diagnostic. Each actor generates three action blocks through the same frozen dynamics and predicted-state feedback used in training. We measure action discrepancy from recorded actions and predicted visual discrepancy from encoded recorded futures, using the fixed action and visual-channel scales of the corresponding training pack. Table 12 averages over windows and all three boundaries. These are normalized errors within a setting; absolute magnitudes should not be compared across predictive representations.

Table 12: **Offline action and predicted-consequence alignment.** Mean normalized MSE over 128 fixed offline validation windows and three boundaries, using one training realization and the survey's epoch-1–10 continuation checkpoints. JACT (Stage 1): Stage 1 pretraining; Local: local-only continuation; Model: model-generated targets. These are recorded-trajectory errors, not simulator success rates.

	Action MSE				Predicted-consequence MSE			
Setting	JACT (Stage 1)	Local	Model	JACT	JACT (Stage 1)	Local	Model	JACT
PointMaze	0.995	0.999	1.009	1.005	0.935	0.963	0.923	0.921
Wall	0.619	0.700	0.629	0.633	0.819	0.920	0.824	0.820
PushT	0.235	0.240	0.218	0.225	0.287	0.292	0.278	0.231
Cube-double	0.443	0.433	0.428	0.428	0.297	0.284	0.272	0.270

On PushT, observed-consequence training reduces predicted-target error more than model-generated targets, while its action error is slightly higher than that control. On Wall, neither error improves over Stage 1. The two metrics therefore do not rank the policies identically, and better latent alignment alone is insufficient to establish improved physical control. In particular, these measurements do not compare predicted and simulated outcomes under the same actor-generated action sequence; they are not an estimate of model accuracy on that sequence.

D.4 Real behavior under matched queries

We replay the first PointMaze development reset group for the initialized policy, GC-IDM, and JACT, recording the simulator observations without modifying policy execution. Initial states, goals, action traces, terminal states, and success labels reproduce the corresponding evaluation records. Relative to Stage 1, JACT succeeds alone on five queries, Stage 1 succeeds alone on one, both succeed on two, and both fail on twelve. Figure 8 shows the lowest-index query from each discordant category. This deterministic selection includes the regression case and does not treat the replay as an independent performance measurement.

D.5 Batch-one deployment latency

We time the unchanged inference path on an NVIDIA RTX 5090: CPU observation transfer, JEPA encoding, actor prediction, action normalization, and return of executable actions to the CPU. Goal features are cached. Simulator stepping and rendering are excluded. For each policy and setting, we cycle over six cached inputs from its first replayed episode, perform 30 warm-up calls, then measure 300 calls with CUDA synchronization. Each call outputs five raw actions. Table 13 reports per-call latency, not latency amortized over the five actions.

The three policies have similar latency, and JACT invokes no predictor. This small, cached-input measurement establishes neither a robust speed ranking across direct policies nor a speedup over online planning.

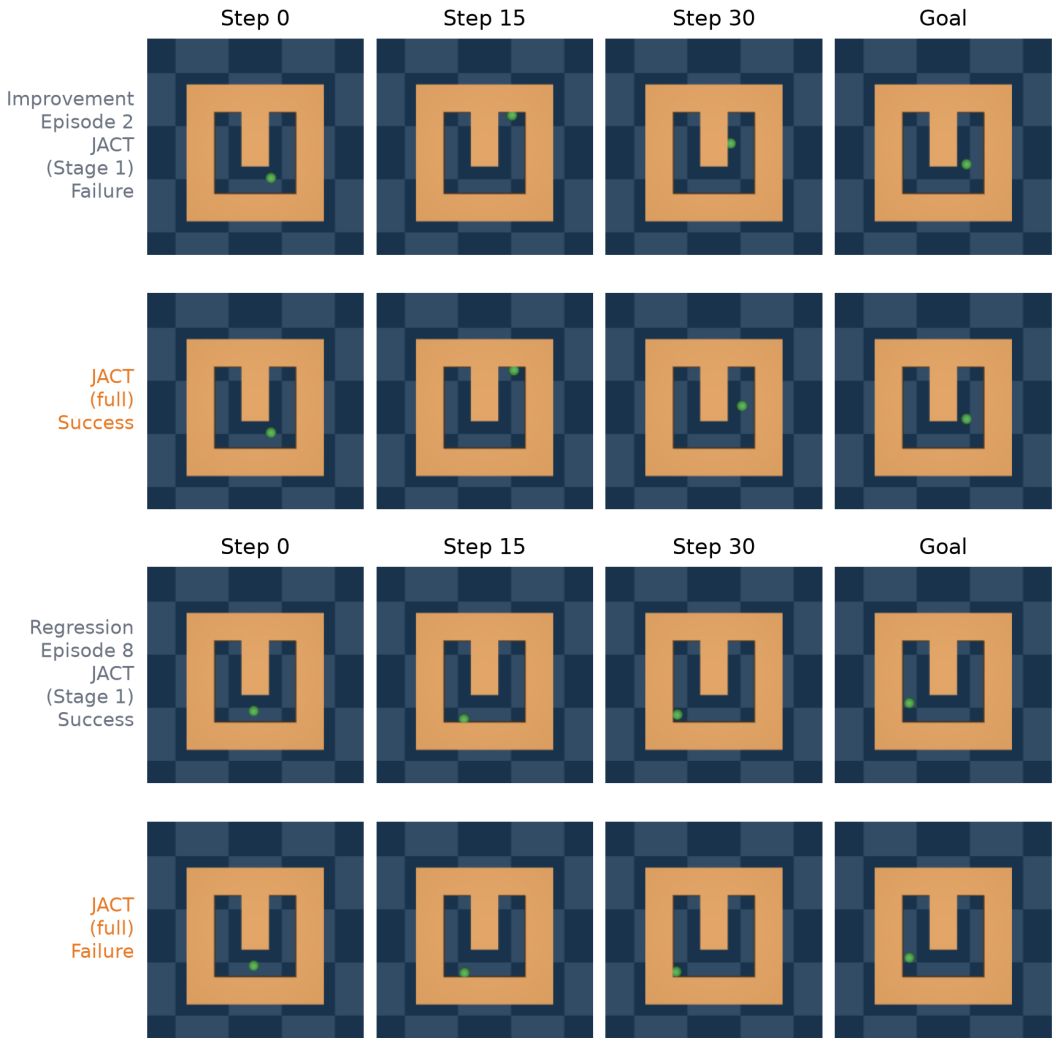

Figure 8: **Real PointMaze observations show both gains and regressions.** Policies share the initial state, goal, and 30-step budget. Columns show raw steps 0, 15, and 30, followed by the goal image; each pair of rows compares JACT (Stage 1) with complete JACT. The top case is the first query on which JACT succeeds and Stage 1 fails; the bottom case is the first with the opposite outcome. Success uses the original terminal criterion. Frames are simulator observations, not decoded model predictions.

Table 13: **Batch-one deployment latency on an RTX 5090.** Milliseconds per five-action decision, including the encoder and CPU output, with the goal encoding cached. Each policy-setting pair uses 30 warm-ups and 300 timed calls over six inputs.

	PointMaze		Wall	
Policy	Median	95th percentile	Median	95th percentile
JACT (Stage 1)	3.820	3.882	3.745	3.791
GC-IDM	3.587	3.637	3.624	3.668
JACT (full)	3.769	3.822	3.815	3.870

Setting	JACT (Stage 1)	GC-IDM	JACT	CEM 128×5	W/L/T
PushT	91	87	90	19	72/1/27
TwoRoom	100	100	100	81	19/0/81
Puzzle-4×4	77	63	87	25	63/1/36
Cube-double	90	89	89	59	31/1/68
PointMaze	18	18	30	99	0/69/31
Wall	35	37	32	65	5/38/57

Table 14: **Direct policies and finite-budget online planning.** Success (%) on the original 100 matched queries per setting. JACT (Stage 1) is the Stage-1 ablation of JACT, not an external method. W/L/T counts queries where JACT alone succeeds, CEM 128×5 alone succeeds, or their binary outcomes agree. Methods share queries and execution budgets within each setting; protocols differ between families.

D.6 Finite-budget online planning

We compare the initialized policy, GC-IDM, and complete JACT with online cross-entropy-method (CEM) planning in six settings. All six controllers share the frozen world model, the original 100 physical start–goal queries, action execution interval, and success criterion within each setting. Direct-policy success labels match the survey in Table 2. Timing uses same-query replays with matching outcomes; action trajectories need not be bitwise identical. These are supplementary evaluations of existing checkpoints on development queries, not additional training runs or new held-out tests. The three fixed CEM budgets are 32 candidates and three iterations, 64 and five, and 128 and five, retaining eight elites. Neither CEM plans nor its outcomes enter JACT training.

LeWM settings: 100 queries each. PushT, TwoRoom, Puzzle-4×4, and Cube-double use all five original reset groups. Goals are observations 25 raw steps ahead; execution is limited to 50 raw steps and stops at success. The native shooting-cost evaluator minimizes goal-embedding mean-squared error. CEM optimizes five consecutive model-action blocks, each containing five distinct raw actions, for a 25-step prediction horizon. Only the first raw action is executed before replanning, matching the direct policies' execution interval. We use a single observed context frame, standardized actions with equivalently standardized environment bounds, and no policy-based initialization, previous-plan warm start, or retrieval. Prediction length remains 25 steps near the execution deadline. This is not a reproduction of the default full-block execution configuration or a tuned upper bound.

PointMaze and Wall: 100 original queries each. We use all five original reset groups from Table 2, preserving the original JEPA-WM checkpoints, initial contexts, and goal states and images. These settings do not use the LeWM checkpoints of the preceding four tasks. The native JEPA-WM planner optimizes up to six five-action blocks, shortening its horizon with the remaining execution budget. One complete five-action block is executed before replanning, matching the direct actors. All methods execute 30 raw steps and use the original terminal-success criterion. Actions have the same physical bounds and clipping as the original policy evaluation. CEM loads no actor and uses no retrieval.

Results. Figure 3 places each controller at its measured success and computation time. Increasing the CEM budget improves success monotonically on TwoRoom and Cube-double, but not on PushT or Puzzle; measured batch-amortized time need not increase monotonically either. All three budgets remain below JACT on the four LeWM settings. Against the largest budget, JACT gains 71/19/62/30 percentage points on PushT/TwoRoom/Puzzle/Cube-double; Table 14 includes both wins and losses. JACT (Stage 1) exceeds JACT by one point on PushT and Cube-double, and all direct policies solve TwoRoom. The clearest consequence-training gain on this subset is on Puzzle, where JACT improves over Stage 1 from 77 to 87%.

On PointMaze, the three CEM budgets reach 79/94/99%, exceeding JACT at 30% and GC-IDM at 18%. Wall exhibits a budget-dependent reversal: CEM reaches 15/56/65%, compared with 32% for JACT and 37% for GC-IDM. Against CEM 128×5, JACT has 0 wins, 69 losses, and 31 ties on PointMaze, and 5 wins, 38 losses, and 57 ties on Wall. These results rule out a uniform performance

advantage of direct control over search on our suite; gains from online computation depend on the task and the planner configuration.

Matched success and latency on navigation queries. The PointMaze and Wall panels in Figure 3 measure batch-one end-to-end decisions on the same 100 queries per setting. CUDA-synchronized timing includes CPU observation preprocessing, input transfer, encoding, actor or planner computation, and CPU action output; simulation and rendering are excluded. Goal encoding is cached and measured separately (approximately 15–17 ms per query, including the initial cold call). Each process makes three non-executed warm-up calls on its first context, restoring random-number-generator state before evaluation. We record all six decisions per episode, giving 600 measurements per method and setting. Every decision commits five raw actions. This rollout measurement is distinct from the cached-input microbenchmark in Table 13.

Mean decision latency is 4.12/4.19 ms for JACT and 3.88/3.96 ms for GC-IDM on PointMaze/Wall. CEM requires approximately 544/546, 1,704/1,706, and 3,410/3,405 ms at the three budgets. Thus the largest budget improves success substantially but takes about 829/813 times as long per decision as JACT in this implementation. This is a measured trade-off at the tested settings, not a lower bound on optimized planning cost or evidence that direct control has comparable success.

Batch-amortized solver time on LeWM settings. For the four LeWM panels, recorded solver time includes context/goal encoding and actor or search computation but excludes outer-policy preprocessing. Total solver time is divided by the number of active environment decisions. Calls include cold starts and variable numbers of active environments, so this quantity is not single-environment latency; nearly coincident CEM timings do not imply equal search work. These measurements are not combined with the navigation end-to-end results to compute a cross-setting speedup.

D.7 Prediction and execution under identical actions

We separately test model accuracy on actor-generated actions. For each of the same 20 PointMaze and Wall development queries, each of four frozen actors generates three five-action blocks through predicted-state feedback, starting from the real initial observation with horizons 30, 25, and 20. We feed the identical physical actions to the frozen predictor and simulator. No real observations are used to replan during this 15-step probe. Actions use the deployment clip $[-1, 1]$; none are clipped in these trials. Thus the experiment isolates prediction versus execution for a fixed plan, rather than evaluating the deployed closed-loop policy.

Every plan is executed twice after the original full seeded reset and state preparation. Initial and goal observations match the existing query records; all 15 repeated frames match exactly, and physical states match within 10^{-6} absolute and relative tolerance. The first action block also matches normal deployment for the initialized policy and JACT. Two-query preflights precede the full diagnostic and are not counted as additional observations. Actor and world-model weights remain unchanged.

Table 15: **Predicted and executed consequences of identical actions.** Mean normalized visual MSE over 20 matched development queries and raw steps 5, 10, and 15. Each actor generates a three-block plan through JEPA; the simulator executes exactly that plan. Pred./exec.: prediction versus execution; pred./goal and exec./goal: discrepancy from the final goal observation. These are diagnostic plan replays, not the normal closed-loop success evaluation.

Setting	Policy training	Pred./exec.	Pred./goal	Exec./goal
PointMaze	JACT (Stage 1)	8.727	9.563	1.835
	Local only	8.750	9.546	1.852
	Model targets	8.706	9.539	1.826
	JACT (full)	8.728	9.537	1.809
Wall	JACT (Stage 1)	0.078	2.682	2.737
	Local only	0.084	2.681	2.741
	Model targets	0.083	2.671	2.701
	JACT (full)	0.086	2.681	2.742

At each block boundary we encode the real simulator image and compare it with the corresponding prediction, using the training partition's fixed channel scales. Goal errors compare against the final goal image, not the intermediate recorded-future labels in Table 12; the query sets and targets therefore differ between the two diagnostics. On PointMaze, the initialized actor's model error is 8.727, versus 1.562 for repeating the initial representation as a prediction of the same executed sequence. The corresponding Wall errors are 0.078 and 1.422. These setting-specific checks caution against treating the frozen model's free rollout as an accurate state trajectory in every environment.

Relative to Stage 1, JACT changes executed-goal MSE by -0.0256 on PointMaze and $+0.0056$ on Wall. Exploratory paired query-bootstrap 95% intervals are $[-0.0811, 0.0239]$ and $[-0.0309, 0.0420]$, respectively (10,000 resamples, averaging the three boundaries within each query). Both include zero. Predicted-goal error improves while executed-goal error worsens on 5/20 PointMaze queries and 3/20 Wall queries (Figure 9). These probes do not establish a uniform improvement in open-loop consequences or model accuracy, and do not replace the closed-loop success evaluation.

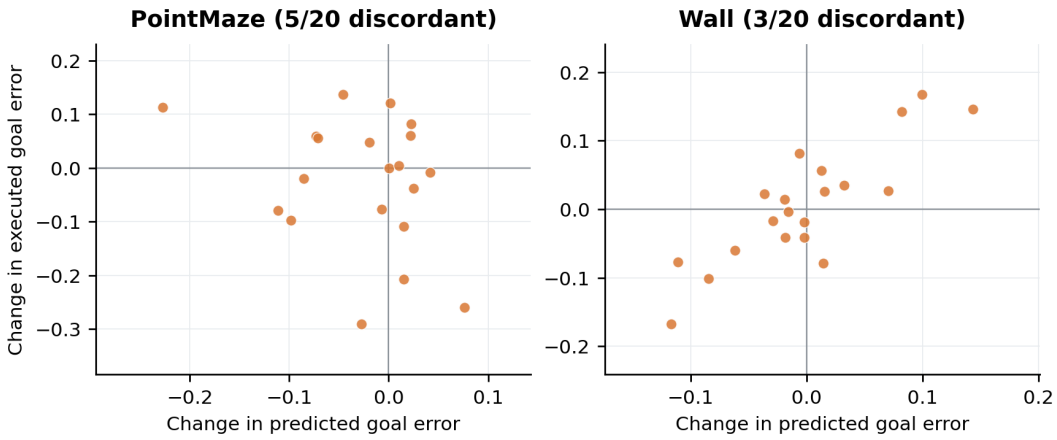

Figure 9: **Predicted improvement need not transfer to execution.** Each point is one of 20 paired queries; changes are complete JACT minus JACT (Stage 1), averaged over raw steps 5, 10, and 15. Negative values denote lower goal-image error. The upper-left quadrant contains plans whose predicted goal error improves but whose real executed goal error worsens. All queries are shown, with separate scales for the two environments. This is an analysis-only imagined-plan replay, not deployment-time planning.

D.8 MANIPULATION TRAJECTORIES AND EARLY COMPLETION

We extend the real-frame replay to PushT and Cube-double, retaining the initialized policy, GC-IDM, and complete JACT. For each setting, the first development group contains 20 queries. All three policies reproduce the archived action-hash sequences, episode lengths, and success labels; the initial policy images and goal images are also identical. These are visualizations of existing trials, not additional independent samples.

Figure 10 illustrates two contrasting successful PushT behaviors from the existing 100-query development cohort: predominantly translating the object, and combining translation with rotation. Both start from visibly displaced object poses. The image goal is the recorded observation at offset $H = 25$, and the execution budget is $B = 50$; displayed stopping times are actual rollout steps. These are qualitative illustrations, not randomly sampled trajectories or evidence of a success-rate advantage. Success uses the goal-conditioned state criterion, including the pusher position, rather than a template-coverage threshold.



Figure 10: **Contrasting PushT behaviors with the same policy.** JACT predominantly translates the object in the top row and combines translation with rotation in the bottom row. Each row shows unmodified observations at steps 0, 10, 20, 30, and the stopping time T, followed by the goal observation. Final object poses substantially overlap the target poses. The green template is the renderer's fixed reference; the rightmost observation specifies the actual goal. These selected successes are illustrative and do not change the aggregate results.

On the first PushT development group, JACT succeeds alone on one query, Stage 1 succeeds alone on two, and both succeed on seventeen. For Cube-double, the two policies both succeed on eighteen queries and both fail on two. Seven queries are successful after a single action for all three policies. Such early completions limit what the high aggregate score can demonstrate about sustained manipulation; the trajectory-offset protocol should not be interpreted as a collection of uniformly difficult, predefined manipulation tasks. Figure 11 shows the lowest-index shared success requiring more than one action from both policies, and the lowest-index shared failure.

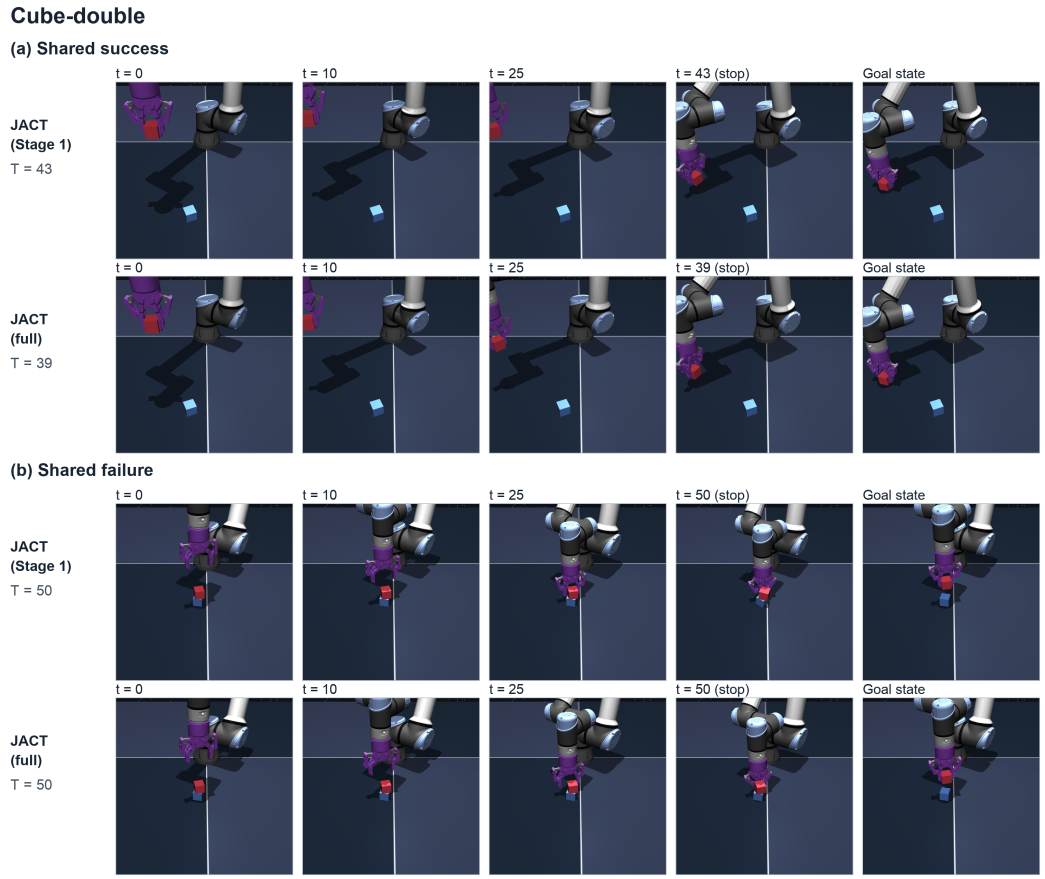

Figure 11: **Cube-double behavior under identical goals.** A shared multi-step success (top) and a shared failure (bottom) compare JACT (Stage 1) with JACT. Columns show the initial state, steps 10 and 25, the stopping state, and the goal state. Saved simulator states are rendered at 512×512 with the original camera, geometry, and lighting, but opaque arm materials for legibility; goal panels use the corresponding dataset goal states. This display-only material change does not alter the action trajectories, physical states, policy observations, or outcomes. Policy images remain 64×64 inputs resized to 224×224.

D.9 SENSITIVITY UNDER THE ORIGINAL TRAINING CONDITIONS

We vary the number of prediction blocks K and consequence weight λ without changing the original Stage 1, training windows, normalization, optimizer, or ten-epoch schedule. All variants use training seed 42 and their own offline validation objective for checkpoint selection. The K sweep fixes $\lambda = 1$; the weight sweep fixes $K = 3$, with $\lambda = 0$ corresponding to local-only continuation. We retain the same windows eligible for the original three-block objective, rather than filtering the training data to accommodate a longer rollout. Evaluation uses the original 100 start–goal queries in each setting.

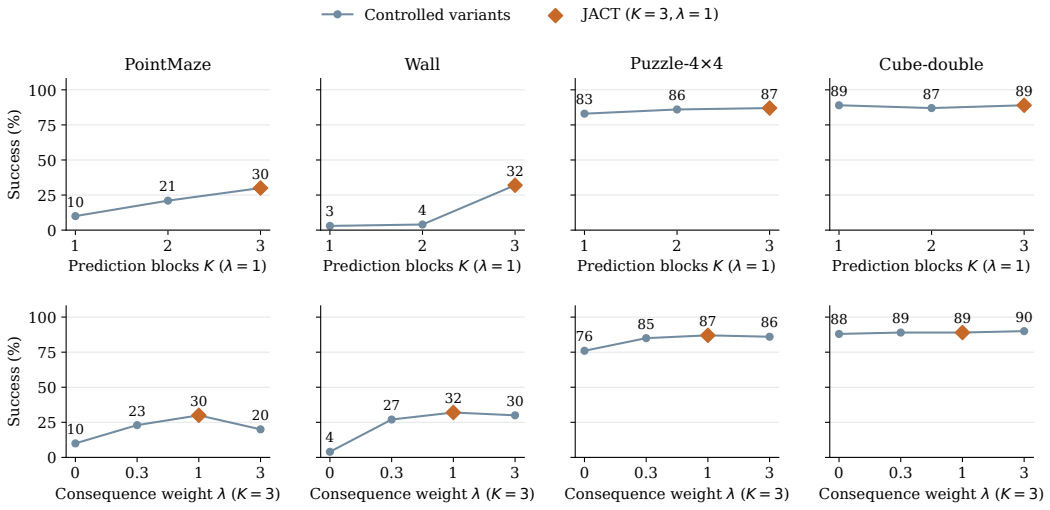

Figure 12: **Sensitivity to rollout length and consequence weight.** Orange diamonds mark $K = 3, \lambda = 1$, matching Table 7. Every point uses one training realization, epoch-1–10 selection, and the original 100 development queries. Top: prediction blocks. Bottom: consequence weights on a categorical axis. Segments only organize discrete configurations.

The full three-block objective achieves 30/32/87/89% on PointMaze/Wall/Puzzle/Cube, compared with 21/4/86/87% for two blocks and 10/3/83/89% for one block. Weight sensitivity is task-dependent: changing λ from 1 to 0.3 or 3 reduces PointMaze success to 23 or 20%, whereas Cube remains at 89 or 90%. Wall retains 27–32% across nonzero weights but drops to 4% under the original local-only recipe. The stronger local-only procedure instead reaches 31% (Appendix C.3); this curve therefore does not establish a large advantage over supervised continuation in general. No per-task best configuration is substituted into the main comparison.

D.10 SENSITIVITY TO GOAL OFFSET AND EXECUTION BUDGET

We examine a fixed policy across recorded-future goals at temporal offsets $H \in \{25, 50, 75, 100\}$. For each task, we train one JACT (Stage 1) actor and one GC-IDM actor for 50 epochs with future-goal offsets up to 100, then train JACT for ten additional epochs from that Stage 1, using $K = 3$, $q = 5$, and consequence weight $\lambda = 1$. The JEPA is frozen throughout. Checkpoints are selected using offline validation and remain unchanged across evaluation offsets; no model is selected separately for an offset in Figure 5.

Within each task, the three methods use the same set of distinct development starts, shared across all goal offsets. Different starts may come from the same source trajectory. We exclude starts that already meet the native success criterion at any of the four goals. These are one-training-seed experiments, not independent training repeats. Increasing H changes the recorded goal and need not increase geometric difficulty monotonically. We report both $B = 2H$, which increases available interaction with the offset, and a fixed $B = 50$ comparison. The $H = 25, B = 50$ cell belongs to both comparisons. The actor's remaining-budget input is clipped to $[1, 100]$ before division by 100, so the longer budgets still include clipped inputs even though all goal offsets lie within the training range.

The effects are task-dependent. On PushT, mean success across the four offsets at $B = 2H$ is 66.25% for both JACT and Stage 1, compared with 62.25% for GC-IDM. At fixed $B = 50$, the respective means are 48.25%, 49.0%, and 45.75%. On Cube-single, they are 93.25%, 92.0%, and 88.0% at $B = 2H$, and 92.25%, 89.75%, and 84.75% at fixed $B = 50$. JACT matches or exceeds both controls in every Cube-single cell, whereas PushT exhibits both gains and regressions. Reacher remains near ceiling (97–100%): JACT and Stage 1 have identical episode outcomes in all seven cells, and GC-IDM differs only by one additional failure at $H = 50, B = 50$. Thus Reacher provides no evidence of a consequence-training advantage over Stage 1. All three methods solve every TwoRoom query at every offset and budget. These within-task averages reuse starts under different goals and budgets, rather than pooling independent trials. For PushT, Cube-single, and TwoRoom, the training goal sampler and partitions differ from the shorter-range survey actors; their $H = 25$ cells are not substituted into the survey. Reacher is the explicit exception: Tables 1 and 2 reuse this study's $H = 25, B = 50$ actors and queries. Cross-protocol comparisons do not isolate the effect of expanding the training horizon.

D.11 COMPUTATION REQUIRED BY CONSEQUENCE TRAINING

Consequence training adds offline computation without changing deployment cost. We measure the second-stage forward and backward passes on an RTX 5090, using the existing batch sizes (32 for navigation and 128 for manipulation), FP32, and fixed training-partition latents resident on the GPU. The initialized actor and frozen predictor are loaded without updating or saving either model. Each variant is measured for three rounds of five warm-up and twenty timed passes; the variant order rotates between rounds. CUDA synchronization brackets each pass, and gradients are cleared before it. Table 16 reports the median over sixty measurements, not a full-run training time.

Table 16: **Cost of the second-stage objective.** Median synchronized forward/backward time in ms, with peak allocated GPU memory in MiB in parentheses. Batches contain cached latents. The unpruned local-loss implementation evaluates the same rollout graph with zero consequence weight; the optimized local loss omits it. No optimizer updates are performed.

Setting	Batch	Optimized local	Unpruned local	JACT
PointMaze	32	3.04 (239)	384.07 (27,349)	388.83 (27,349)
Wall	32	3.07 (239)	384.85 (27,349)	390.33 (27,349)
PushT	128	1.86 (104)	46.34 (393)	46.34 (393)
Cube-double	128	2.45 (104)	46.44 (393)	46.49 (393)

We distinguish an optimized local-supervision computation from the unpruned implementation used by the equal-update continuation control. The latter still evaluates and backpropagates through the rollout graph when its consequence coefficient is zero, and consequently has nearly the same measured cost as JACT. Omitting that unused graph produces the same local loss and parameter gradients in an evaluation-mode check (maximum measured gradient difference zero), but is much cheaper. This distinction prevents a redundant control implementation from making consequence training appear cost-free. The optimized computation is a timing control, not a newly trained policy.

These measurements exclude image encoding, data transfer and loading, optimizer updates, checkpoint validation, and file I/O; no full-training GPU-hours are inferred. Peak allocated memory includes the resident frozen model even for the optimized local computation and excludes unused reserved memory. Model weights and buffers are unchanged after all measurements. The cost is paid during offline training, whereas deployment continues to use the encoder and actor without predictor rollouts.

D.12 SUPERVISED CONTINUATION WITH MATCHED TRAINING TIME

Equal optimizer-update counts do not equate training cost. We therefore give optimized action-and-waypoint supervision the measured active-training-loop time of the complete consequence-training stage. Both start from the same Stage 1 pretraining and use the original training windows, batch size, seed 42, AdamW settings, and gradient clipping. The control omits the unused predictor graph; its local loss and parameter gradients are checked against the zero-weight objective before training. It can consequently perform many more supervised updates within the same time budget.

The budget measures CUDA-synchronized loop time, including batch transfer, forward and backward passes, and optimizer updates, but excluding model loading, warm-up, offline validation, and saving. We divide this budget into ten equal-time segments with cosine learning-rate scheduling and one offline validation check per segment, selecting by supervised validation loss. Thus validation opportunities match the ten-epoch reference, but scheduling is time-based rather than epoch-based. All measurements use RTX 5090 GPUs; Puzzle and Cube use a different physical card from their reference timing, whereas the other five settings use the reference card. These are measured training-time controls, not matched FLOP counts or total-job runtimes.

Figure 13 compares all seven settings; Table 17 provides training budgets and per-setting results.

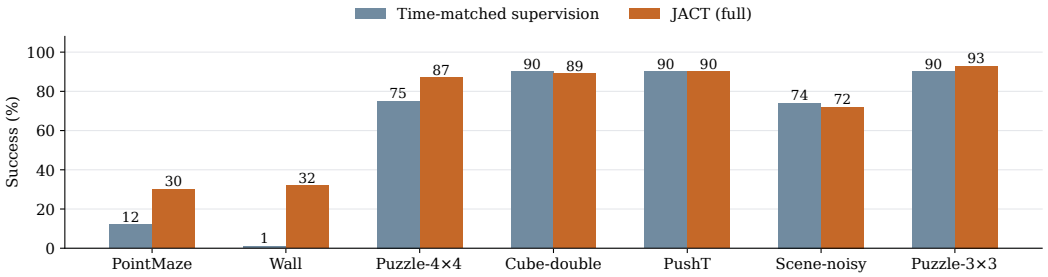

Figure 13: **Consequence training versus time-matched supervision.** Success on the original 100 queries per setting. JACT retains the survey's epoch-1–10 checkpoints. Time-matched supervision uses the same active-training-loop budget and selects among ten elapsed-time checkpoints; neither continuation includes Stage 1. Schedules and selection objectives differ.

	Training-loop time (s)		Optimizer updates		
Setting	JACT budget	Supervised	JACT	Supervised	W/L/T
PointMaze	516.76	516.77	1,280	95,735	24/6/70
Wall	518.54	518.54	1,280	96,680	31/0/69
Puzzle-4×4	144.10	144.10	2,560	33,206	12/0/88
Cube-double	147.02	147.03	2,560	34,151	1/2/97
PushT	125.94	125.94	2,560	28,977	3/3/94
Scene-noisy	143.72	143.72	2,560	34,333	4/6/90
Puzzle-3×3	146.10	146.10	2,560	34,525	7/4/89

Table 17: **Training-time allocation and paired outcomes.** The optimized supervised continuation uses the same measured active-training-loop budget as JACT. W/L/T compares JACT with this control on 100 original queries. Loading, warm-up, validation and checkpoint saving are excluded from both timing budgets. Updates count the complete training run, not only the selected checkpoint's prefix.

JACT improves over time-matched supervision by 18 points on PointMaze, 31 on Wall, 12 on Puzzle-4×4, and 3 on Puzzle-3×3. It ties on PushT and loses one point on Cube-double and two on Scene-noisy. The seven-setting means are 70.43% and 61.71%, respectively. These results describe the fixed time-matched recipe, not the best supervised baseline: the stronger continuation in Appendix C.3 reaches 31% on Wall, nearly matching JACT at 32%, with both below Stage 1 (35%) and GC-IDM (37%). Each time-matched run selects its first validation checkpoint; the reported

update counts describe the full training runs, not the prefixes ending at the selected checkpoints. Scheduling and stochastic training differences also prevent a FLOP-isolated causal interpretation.

D.13 GRADIENTS THROUGH PREDICTED-STATE FEEDBACK

The recorded-state control changes what the actor observes during training. Here we instead preserve predicted-state inputs and stop only their feedback gradients into subsequent actor calls. For identical weights, the actor actions, model predictions, and losses are numerically unchanged; the derivative of the actor input with respect to earlier predictions is removed. This tests the feedback path in Equation (6), not the presence of the consequence loss or the value of predicted contexts.

On PointMaze and Wall, we detach the predicted visual context before pooling it for the next actor call; these actors receive no action-history input. On Puzzle and Cube, we detach the predicted state context at JEPA block boundaries. Each block contains five one-step actor calls, and their within-block waypoint feedback remains differentiable. In both interfaces, gradients through JEPA's internal state history and action prefixes are retained. This is not detachment of the entire predictor or of every within-block feedback edge. Checks at identical weights confirm unchanged forward outputs, identical gradients for $K = 1$, and different actor gradients for $K = 3$.

The stopped-feedback variant shares the complete method's training configuration. Each setting retains its original Stage 1, training windows, normalization, optimizer schedule, $K = 3$, and $\lambda = 1$. Navigation uses 4,096 training and 512 validation windows, batch size 32, and 1,280 updates; Puzzle and Cube use 32,768 and 4,096 windows, batch size 128, and 2,560 updates. All runs use training seed 42 and ten epochs, selecting checkpoints by their own offline validation objective. Evaluation reuses the original 100 start–goal queries per setting, with matching initial contexts and goal observations.

Table 18: **Gradient-only feedback ablation.** Success (%) $\pm$ reset-group SD on the original 100 development queries, one training run, and epoch-1–10 selection for both variants. W/L/T counts complete JACT wins, losses, and ties relative to stopped feedback.

Setting	**JACT (full)**	Stopped feedback gradient	W/L/T
PointMaze	30 ± 7.1	25 ± 11.7	13/8/79
Wall	32 ± 11.5	28 ± 10.4	5/1/94
Puzzle-4×4	87 ± 4.5	86 ± 5.5	1/0/99
Cube-double	89 ± 6.5	89 ± 6.5	0/0/100

Under the fixed epoch-1–10 recipe, complete JACT reaches 30/32/87/89%, compared with 25/28/86/89% after stopping feedback gradients (Table 7); the means are 59.50% and 57.00%. Table 18 retains these fixed-recipe paired outcomes. With Stage 1 eligible, both Wall variants select the same Stage 1 and reach 35%, so this setting provides no evidence of a feedback-gradient benefit. PointMaze still includes both wins and losses; Puzzle differs by one success; Cube has identical labels. These results distinguish predicted-context conditioning from differentiating through it, but do not establish a universal benefit or stability across independent training seeds.

D.14 Learning with fewer actor-training episodes

We restrict actor training to nested subsets containing 10, 25, 50, or 100% of the original training episodes, while keeping the pretrained JEPA fixed. Each reduced subset is used to train both JACT (Stage 1) and GC-IDM from scratch; JACT then starts from the Stage 1 trained on that same subset, never from a full-data actor. Episode selection is deterministic and independent of policy outcomes. Puzzle-3×3 uses 57/142/284 of 567 training episodes; the other settings use 90/225/450 of 900. Consequence training retains only the original windows belonging to the selected episodes and recomputes its outcome scale from those windows. Actor normalization uses only selected training data, while JEPA normalization remains unchanged.

All fractions retain their original optimization recipes: fifty epochs for Stage 1 and GC-IDM, followed by ten epochs for JACT with $K = 3$ and $\lambda = 1$. Validation episode pools are fixed; consequence-validation windows and goals are unchanged. Supervised training retains its original stochastic future-goal sampler. Each configuration uses training seed 42 and the same 100 evaluation queries, a goal offset of 25 steps, and a 50-step execution budget. Epochs rather than optimizer updates are fixed, so smaller subsets also receive fewer updates. The experiment measures actor-stage data efficiency conditional on a full-data pretrained model, not end-to-end data efficiency.

Figure 4 shows the clearest benefits on puzzles. On Puzzle-4×4, JACT reaches 81% with 25% of the actor data, versus 73% for same-subset Stage 1 and 77% for full-data Stage 1. On Puzzle-3×3, the corresponding gains over Stage 1 are 4/3/1/1 points across the four fractions. Scene-noisy changes by $+2/-1/+1/+2$ points and Cube-double by $-2/+1/-1/-1$: neither exhibits a consistent large gain. Nor does JACT uniformly outperform GC-IDM, which leads at 25% on Cube-double and at 100% on Scene-noisy. These results support task-dependent benefits from consequence training rather than a universal low-data advantage.

Does additional supervised optimization explain the gains? At 10/25/50%, we additionally continue each same-subset Stage 1 with only its supervised action and waypoint losses ($\lambda = 0$). This control matches JACT in training windows, batch size (128), ten continuation epochs, optimizer and learning-rate schedule, and training seed. It matches update counts within each fraction, not training time or updates across fractions. We retain the original computation path with zero consequence weight to preserve random-number consumption. Checkpoints are chosen by the corresponding offline validation objective, never by control success.

Relative to this control, JACT gains 6/10/8 points on Puzzle-4×4, 6/0/0 on Puzzle-3×3, 4/0/3 on Cube-double, and 3/1/1 on Scene-noisy. Across these twelve task–fraction pairs, mean success is 77.08% versus 73.58%, with nine gains and three ties. Thus equal additional supervised updates do not fully explain the observed gains. This comparison uses one training seed, however, and reuses the same queries across fractions; the twelve cells are not independent query samples, nor does this result establish an advantage under equal training time.

D.15 Do consequence gradients improve real execution?

Consequence matching supplies an action-space direction through the frozen predictor (Appendix E). We test whether small steps in that direction improve actual consequences, rather than merely lowering predicted error. This diagnostic uses the first twenty training trajectories in fixed episode order for each of PointMaze and Wall, at their initial states; it does not use the development queries in Appendix D.7.

We first replay 25 recorded actions in the simulator. The resulting observations at steps 5, 10, and 15 define intermediate visual targets, and the step-25 image defines the actor goal. A frozen JACT (Stage 1) actor then generates a fifteen-action plan through three blocks of predicted-state feedback, with remaining horizons 25, 20, and 15. Both the actor and JEPA remain unchanged. Starting from this same plan, we compare the negative consequence-loss gradient, the negative action-MSE gradient toward the first fifteen recorded actions, and a fixed random direction. All directions are computed in action coordinates normalized by JEPA's training-set action standard deviations and rescaled to unit root-mean-square magnitude. We perturb the plan by RMS steps 0.01, 0.03, and 0.10, then execute it open-loop without replanning.

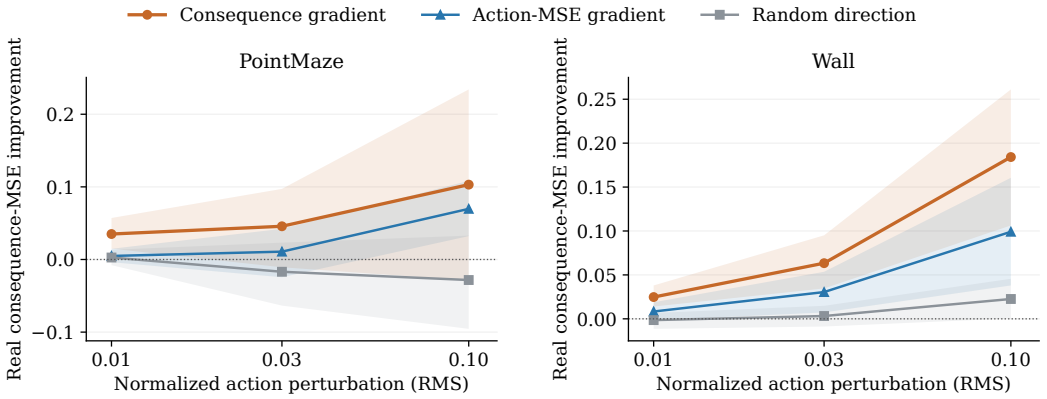

Figure 14: **Local consequence gradients can improve real execution.** Positive values denote a reduction in normalized visual MSE to replayed recorded-action targets, averaged over steps 5, 10, and 15. Lines average twenty source trajectories; shading gives exploratory 95% episode-bootstrap intervals (10,000 resamples), not training-seed variation. Every tested step size is shown. Panels use task-specific scales and are not pooled. This is a diagnostic of action perturbations at a frozen initialized actor, not the deployed JACT policy or a success-rate comparison.

Before every replay we restore the trajectory's physical state and, for Wall, wall and door locations. Each reference and candidate sequence is executed twice: frames agree exactly, and physical states and encoded observations agree within 10^{-6}. No tested action is clipped. The error uses the same frozen encoder and channel scales for all conditions, so improvements concern visual representation error, not necessarily physical goal attainment.

At RMS step 0.01, the consequence direction reduces predicted error on all twenty trajectories in each task, but real error on only 15/20 PointMaze and 18/20 Wall trajectories. Mean real-error reductions are 0.0351 and 0.0248, compared with 0.0049 and 0.0083 for the action-MSE direction. Figure 14 shows positive mean real improvement for the consequence direction at all three tested magnitudes, with substantial trajectory variability. These observations support local transfer of the predictive gradient, while also exposing imperfect transfer to execution. Because the probe uses training trajectories and fixed action-space perturbations, it neither establishes held-out policy improvement nor accounts for the Wall regression of the trained actor.

E ACTION-SPACE INTERPRETATION OF CONSEQUENCE MATCHING

Let $R(A) \in \mathbb{R}^m$ vectorize the visual predictions along a rollout, y the observed targets, and D the diagonal matrix of inverse-squared channel scales, repeated over time and spatial positions. Then

$$\ell_{\text{con}}(A) = \frac{1}{m}(R(A) - y)^\top D(R(A) - y), \qquad \nabla_A \ell_{\text{con}} = \frac{2}{m} J_R(A)^\top D(R(A) - y). \tag{12}$$

Thus consequence matching weights discrepancies by their action sensitivity under the predictor, rather than by action distance alone. It cannot distinguish action directions to which the model is locally insensitive.

For twice continuously differentiable R, let $e_D = R(A_D) - y$ at recorded actions A_D. A second-order expansion gives

$$\ell_{\text{con}}(A_D + \delta) = \ell_{\text{con}}(A_D) + \frac{2}{m}\delta^\top J_D^\top D e_D \tag{13}$$

$$+ \frac{1}{m}\delta^\top \left[J_D^\top D J_D + \sum_{i=1}^{m} (De_D)_i \nabla_A^2 R_i(A_D) \right] \delta + o(\|\delta\|^2). \tag{14}$$

If $e_D = 0$, the quadratic term is a positive-semidefinite pullback metric. Otherwise, residual-dependent terms can move actions to compensate model error. Model-generated targets $y = R(A_D)$

ensure zero residual at recorded actions, not better control. Composing with the actor gives $\nabla_\theta \ell = (\partial A_\theta / \partial \theta)^\top \nabla_A \ell$, including the feedback paths in Equation (6). These local identities do not guarantee policy improvement.