DSSF: ROBUST LATENT STATE CONSTRUCTION FOR JEPA-BASED VISUAL PLANNING

Anonymous authors
Paper under double-blind review

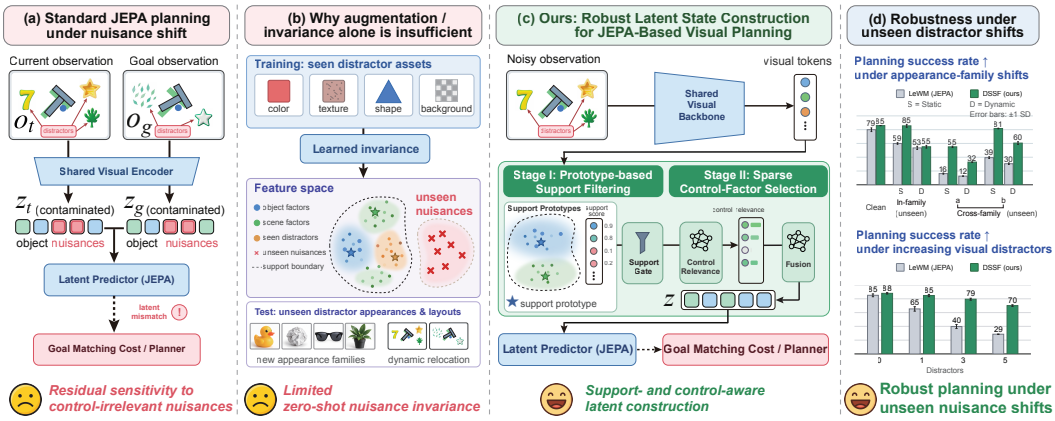


Figure 1: Method motivation. (a) In standard JEPA-based visual planning, nuisances in the current and goal observations may be encoded into the latent states, thereby interfering with latent prediction and goal matching. (b) Data augmentation typically adapts the model to perturbations observed during training, but may generalize poorly to unseen distractor appearances. (c) To address this limitation, we propose DSSF, which performs prototype-based support filtering followed by sparse control-factor selection before the JEPA latent predictor, constructing a more stable and control-relevant latent state from nuisance-corrupted observations. (d) The resulting representations enable more robust planning under increasing numbers of zero-shot visual distractors.

ABSTRACT

Latent world models based on the Joint-Embedding Predictive Architecture provide an efficient approach to dynamics modeling for visual planning by predicting future representations rather than reconstructing pixels. However, latent prediction primarily optimizes the predictability of visual information and does not guarantee that the retained factors are relevant to control. Consequently, task-irrelevant backgrounds and visual distractors may be encoded into the latent state, perturbing state prediction and goal matching when previously unseen visual distractors occur at test time. We propose DSSF, a two-stage latent-state construction method for improving planning robustness under unseen visual distractors. In the first stage, learnable prototypes model the feature-space support of non-distractor visual factors and filter out local factors that are not supported by the training distribution. In the second stage, DSSF sparsely selects information relevant to action-conditioned dynamics from the retained factors to construct a compact latent state. In this way, DSSF separately models whether a visual factor is supported by the training distribution and whether it is relevant to control, thereby suppressing both unseen distractors and in-distribution yet control-irrelevant information. We evaluate unseen distractor instances and cross-family distractors to systematically assess planning robustness under increasingly severe shifts in distractor appearance and count. Across multiple continuous-control tasks, DSSF better preserves clean-setting planning performance and consistently outperforms strong baselines under unseen distractors and cross-family distractor-appearance shifts, indicating that its robustness does not rely on memorizing the appearances of training distractors.

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

1 INTRODUCTION

Predicting action consequences from high-dimensional visual observations is fundamental to embodied planning and control. World models allow agents to evaluate candidate action sequences through learned dynamics without repeated interaction with the environment (Hafner et al., 2019; Schrittwieser et al., 2020; Hafner et al., 2025). More recently, large-scale world foundation models such as NVIDIA Cosmos have extended this paradigm toward reusable physical simulation for robotics and autonomous systems (NVIDIA et al., 2025). However, pixel-space prediction must model decision-irrelevant details such as textures, lighting, and backgrounds. Learning dynamics in a compact latent space reduces this burden and focuses model capacity on useful state information.

The Joint-Embedding Predictive Architecture (JEPA) provides a reconstruction-free approach to world modeling by predicting future representations rather than pixels (Assran et al., 2023). Recent methods extend this idea to action-conditioned latent dynamics and planning through latent rollouts and goal matching (Zhou et al., 2025; Sobal et al., 2025). LeWorldModel further shows that a visual encoder and dynamics predictor can be trained end-to-end from raw pixels using latent prediction and representation regularization (Maes et al., 2026). Yet avoiding pixel reconstruction does not ensure that latent representations discard task-irrelevant information. Predictive objectives reward predictability, but predictability does not imply control relevance. Stable backgrounds, textures, and visual distractors may therefore enter the latent state even when they are unrelated to control. When such factors change at test time, they can perturb current and goal representations, affecting latent dynamics and planning (Toso et al., 2026). Learning stable and control-relevant representations under unseen visual nuisance shifts thus remains a key challenge for latent world models.

Data augmentation can improve robustness to visual variations seen during training (Yarats et al., 2021; Ha et al., 2023), but it does not explicitly determine whether a new visual factor should enter the dynamics representation. Conversely, useful invariance must preserve task-relevant dynamical information (Sun et al., 2024; Toso et al., 2026). We therefore study nuisance shifts where visual appearance changes while the underlying task state, action space, and environment dynamics remain unchanged. Under this setting, we decompose latent state construction into two questions: whether a visual factor is supported by the non-distractor training distribution, and whether a supported factor is relevant to action-conditioned dynamics.

Based on this decomposition, we propose DSSF (Distribution-Supported Sparse Control Factors). In the first stage, learnable Support prototypes model the multimodal structure of non-distractor visual factors, and a Support Gate filters local factors with insufficient training-distribution support. Here, support approximates the feature-space support of the training data rather than a calibrated density or general OOD criterion. In the second stage, DSSF applies sparse Control-Factor Selection to the retained factors and aggregates the selected ones into a latent state for dynamics prediction and planning. DSSF thus separately handles unsupported unseen factors and supported but control-irrelevant information. DSSF is an encoder-side state construction module compatible with the LeWM predictor and latent planning pipeline. It is trained end-to-end with a single augmented visual input stream using latent prediction, representation regularization, support supervision, and auxiliary action recovery. To test whether DSSF learns a transferable factor-selection mechanism rather than memorizing training distractor appearances, we evaluate generalization from held-out in-family instances to unseen distractor-appearance families. Our main contributions are:

- We show that visual predictability in latent predictive world models does not necessarily imply control relevance. We therefore decompose latent state construction under unseen visual distractors into two problems: determining whether a visual factor is supported by the training distribution and identifying control-relevant factors among the supported ones.
- We propose DSSF, a two-stage latent state construction method that first filters local factors lacking training-distribution support with learnable Support prototypes, and then applies sparse Control-Factor Selection to construct a compact action-conditioned latent state.
- We evaluate DSSF across multiple continuous-control tasks and zero-shot generalization across unseen distractor appearances from unseen in-family instances to cross-family distractors. DSSF preserves clean-setting planning performance while reducing degradation under increasing visual interference and improving latent prediction and planning robustness under unseen distractors.

2 Related Work

JEPA-based World Models for Planning. World models support planning by learning action-conditioned dynamics and rolling out imagined futures Wang et al. (2026); Im et al. (2026). Compared with generative world models that reconstruct future observations, Joint-Embedding Predictive Architectures (JEPAs) learn dynamics in compact latent spaces by predicting future representations. DINO-WM learns action-conditioned dynamics over frozen DINOv2 features, PLDM jointly learns representations and predictors with VICReg-style regularization, temporal constraints, and inverse dynamics, and LeWorldModel (LeWM) further simplifies stable end-to-end training using next-embedding prediction and SIGReg (Zhou et al., 2025; Sobal et al., 2025; Maes et al., 2026). Recent work further improves JEPA world models through representation regularization and action-sensitive latent dynamics (Zhao et al., 2026; Zhang et al., 2026). These advances show that reconstruction-free latent dynamics can support effective visual planning, but they primarily address representation collapse and action sensitivity rather than how unseen visual nuisances affect latent states and downstream planning.

Robust Visual Representations under Nuisance Shifts. Robustness to background changes and task-irrelevant distractors has been widely studied in visual control. Existing methods improve robustness through data augmentation, contrastive learning, bisimulation, masking, or task-relevant attention (Ha et al., 2023; Sun et al., 2024). Recent work also introduces control-relevant invariance into JEPA-based planning to suppress irrelevant backgrounds and distractors (Toso et al., 2026). DSSF instead approaches this problem from latent state construction and separates two questions: whether a local visual factor is supported by the training visual distribution, and whether a supported factor is relevant to control. Based on this decomposition, DSSF first filters visual factors lacking training-distribution support and then sparsely selects factors relevant to action-conditioned dynamics, improving robustness to unseen nuisance shifts.

3 Method

We propose DSSF, a two-stage latent-state construction method for JEPA-based planning under unseen visual distractors. As shown in Fig. 2, DSSF first filters local visual factors unsupported by the training distribution and then sparsely selects control-relevant factors to construct the latent state used by action-conditioned dynamics. The two stages separate distributional support from control relevance: a frequently observed static background may be well supported yet irrelevant to dynamics, whereas an unseen distractor may lack training-distribution support.

3.1 Problem Setup and Framework Overview

We consider fully offline, reward-free learning of a visual world model. The training data consist of trajectories of visual observations and actions, where o_t denotes the visual observation at time t, and a_t denotes the action that transitions the environment from o_t to o_{t+1}. We focus on observation-level nuisance shifts, under which the appearance of the visual observation changes while the underlying task state, action space, and environment dynamics remain unchanged. We denote an observation affected by a visual distractor as $\tilde{o}_t = A(o_t; d)$, where d denotes the nuisance factor. A standard latent world model encodes the visual observation into a latent state z_t and predicts the next state from the current state and action:

$$z_t = E(\tilde{o}_t), \qquad \hat{z}_{t+1} = P(z_t, a_t). \tag{1}$$

Although this reconstruction-free formulation avoids pixel-level prediction of future observations, latent prediction alone does not specify which visual information should enter the state representation. Predictable backgrounds, textures, and even task-irrelevant distractors may therefore be retained and, when they change at test time, perturb both latent dynamics and goal matching.

DSSF redesigns the construction of the latent state from visual observations. As shown in Fig. 2, a shared visual backbone extracts spatially aligned shallow and deep features. The shallow features preserve more local appearance information and are used to estimate whether each spatial location is supported by the training visual distribution, whereas the deep features are used for subsequent latent-state construction. In Stage I, the shallow features produce a spatial Support Gate, which

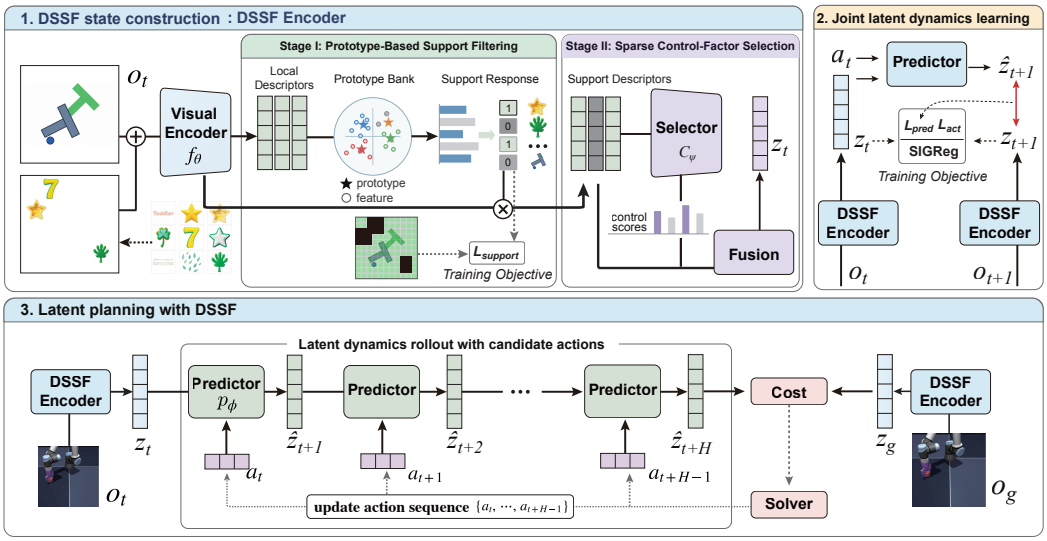


Figure 2: **Overview of DSSF.** DSSF constructs a compact latent state through two-stage selection: Prototype-Based Support Filtering identifies spatial locations supported by the training visual distribution, while Sparse Control-Factor Selection selects and fuses a small set of Control factors into z_t. During training, the encoder and predictor are jointly trained with support, prediction, SIGReg, and auxiliary action objectives. During planning, CEM optimizes candidate action sequences through latent rollout and terminal goal matching.

filters the deep features at the corresponding locations. The retained deep features are defined as Support descriptors. Stage II then further selects Control factors from these Support descriptors.

3.2 DSSF State Construction

3.2.1 Stage I: Prototype-Based Support Filtering.

The objective of this stage is to determine whether the local visual content at each location is supported by the training visual distribution. Let $p_{t,i} \in \mathbb{R}^{d_s}$ denote the shallow feature extracted by the backbone at location i. To represent the diverse modes of visual factors in the training data, we maintain a Support prototype bank $\mathcal{S} = \{s_k\}_{k=1}^{K_s}$ comprising K_s learnable vectors, where $s_k \in \mathbb{R}^{d_s}$. For each shallow feature $p_{t,i}$, we compute its cosine similarity with every Support prototype and take the maximum similarity as the Support response at that location:

$$R_{t,i} = \max_{k=1,\dots,K_s} \cos(p_{t,i}, s_k). \tag{2}$$

This multi-prototype design allows individual prototypes to cover different local modes of the training visual distribution. A local feature produces a high Support response as long as it matches at least one such mode, whereas its response decreases when it matches none of the prototypes. We normalize the Support response to $\bar{R}_{t,i} \in [0, 1]$ and obtain a binary spatial Support Gate using a threshold τ: $G_{t,i} = \mathbf{1}[\bar{R}_{t,i} \geq \tau]$. Here, $R_{t,i}$ denotes a prototype-based support score relative to the non-distractor visual patterns observed during training. Its role is to determine whether the local content can be supported by the non-distractor visual patterns learned from the training data.

Learning the Support Prototypes. To learn this support structure, we randomly add synthetic distractors to the original visual observations during training, with the augmentation procedure simultaneously providing the corresponding spatial masks. The distractor mask is used only to indicate which regions should be learned as non-distractor visual patterns and which regions are affected by a distractor. Because convolutional features have nonzero receptive fields, features near a distractor boundary may be influenced by both normal and distractor regions. We therefore partition the spatial locations into three subsets: regions sufficiently far from the distractor form the positive

set $\mathcal{P}$, regions directly affected by the distractor form the negative set $\mathcal{N}$, and ambiguous locations near the boundary form $\mathcal{I}$ and are excluded from support supervision. The support loss encourages high Support responses in positive regions while suppressing the responses in negative regions:

$$\mathcal{L}_{\text{sup}} = \frac{1}{2}\,\text{BCE}(\bar{R}_{\mathcal{P}}, 1) + \frac{1}{2}\,\text{BCE}(\bar{R}_{\mathcal{N}}, 0). \tag{3}$$

The prototype bank therefore learns the feature-space support of non-distractor visual factors in the training data, rather than the categories of distractors encountered during training. At test time, neither a distractor mask nor prior knowledge of the distractor category or appearance is required.

The Support Gate does not directly constitute the final state; instead, it filters the deep features aligned with the shallow feature space. For the deep feature $f_{t,i}$ at location i, we add a two-dimensional positional encoding q_i and use $G_{t,i}$ to determine whether the feature proceeds to the next stage. All deep features that pass the Support Gate constitute the Support descriptors of the current observation. However, being supported by the training distribution does not necessarily make a feature relevant to control. Stable backgrounds and static textures, for example, may also receive high Support responses. DSSF therefore performs a second stage of sparse selection.

3.2.2 Stage II: Sparse Control-Factor Selection

Stage II addresses the second question: which information among the Support descriptors should enter the latent state used for action-conditioned dynamics? A shared Control-Factor Selector assigns a Control score $c_{t,i}$ to each Support descriptor. Without explicit semantic supervision, the score is learned end-to-end through the latent-dynamics learning and auxiliary action-recovery objectives, reflecting each visual factor's priority for state construction. To prevent all information that passes the Support Gate from entering the latent state, we retain only the K_c Support descriptors with the highest Control scores. These top-K_c selected features are defined as Control factors.

Let H_t denote the index set of the selected Control factors. After normalizing their Control scores into weights $\alpha_{t,i}$, we construct the final latent state as $z_t = \text{Projector}\big(W_o \sum_{i \in H_t} \alpha_{t,i} f_{t,i}\big)$. The Control scores therefore determine both which Support descriptors are selected as Control factors and how much each selected factor contributes to the final state. Taken together, the two stages play distinct but complementary roles. Stage I preferentially removes information that lacks support under the training visual distribution, keeping the input to the Control-Factor Selector within the supported region as much as possible. Stage II further removes supported but redundant factors that contribute little to action-conditioned dynamics. Through this two-stage selection, DSSF compresses high-dimensional visual observations into a more compact latent state.

3.3 Latent Dynamics Learning

The latent state constructed by DSSF is used directly for action-conditioned dynamics learning. Given adjacent training observations $\tilde{o}_t$ and $\tilde{o}_{t+1}$, the shared DSSF encoder produces z_t and z_{t+1}, respectively. The action-conditioned predictor estimates the next state from the current latent state and action as $\hat{z}_{t+1} = P_\phi(z_t, a_t)$. The latent prediction loss constrains the predicted state to match the latent state encoded from the actual next observation: $\mathcal{L}_{\text{pred}} = \|\hat{z}_{t+1} - z_{t+1}\|_2^2$. Following LeWM (Maes et al., 2026), we also apply SIGReg to the final latent representation to maintain representational diversity and prevent representation collapse. However, future latent prediction alone may still bias the state representation toward easily predictable static information. We therefore introduce an auxiliary action-recovery objective used only during training. Given adjacent latent states z_t and z_{t+1}, an action decoder D_ω recovers the ground-truth action a_t responsible for the state transition. The corresponding action-recovery loss is

$$\mathcal{L}_{\text{act}} = \|D_\omega([z_t; z_{t+1}]) - a_t\|_2^2\,. \tag{4}$$

This objective does not imply that action recoverability is strictly equivalent to control relevance. Instead, we use it as an additional training signal that encourages the final latent state to retain information capable of distinguishing transitions between adjacent states. The action decoder is used only during training and does not participate in latent rollout or planning at test time.

The complete model is jointly optimized using

$$\mathcal{L} = \mathcal{L}_{\text{pred}} + \lambda_{\text{sig}}\mathcal{L}_{\text{SIGReg}} + \lambda_{\text{sup}}\mathcal{L}_{\text{sup}} + \lambda_{\text{act}}\mathcal{L}_{\text{act}}. \tag{5}$$

Here, $\mathcal{L}_{\text{pred}}$ learns the action-conditioned latent dynamics, $\mathcal{L}_{\text{SIGReg}}$ maintains representational diversity, $\mathcal{L}_{\text{sup}}$ learns the support structure of the training visual distribution, and $\mathcal{L}_{\text{act}}$ provides an auxiliary signal for state construction that is related to dynamic changes. Details of the gradient estimator for discrete selection and the training configuration are provided in Appendix H.

3.4 Latent Planning with DSSF

Once trained, DSSF changes only how the latent state is constructed from real visual observations before being passed to the latent dynamics; it leaves the subsequent latent-planning pipeline unchanged. Given a current visual observation o_t and a goal image o_g, the same DSSF encoder maps them to the current state $z_t = E_{\text{DSSF}}(o_t)$ and goal state $z_g = E_{\text{DSSF}}(o_g)$, respectively. For a candidate action sequence $A_{t:t+H-1} = \{a_t, \ldots, a_{t+H-1}\}$, the predictor performs an autoregressive rollout from z_t in latent space: $\hat{z}_{t+k+1} = P_\phi(\hat{z}_{t+k}, a_{t+k}), \hat{z}_t = z_t$. Each candidate action sequence is evaluated by the distance between its predicted terminal state and the latent goal state: $J(A_{t:t+H-1}) = \|\hat{z}_{t+H} - z_g\|_2^2$. We then optimize the candidate action sequences using the Cross-Entropy Method and adopt receding-horizon model predictive control: after executing the action selected by the current optimization step, the agent obtains a new visual observation and repeats the planning procedure above.

4 Experiments

Our experiments address four questions. First, can DSSF improve the robustness of latent planning to visual distractors across tasks? Second, does this robustness generalize beyond training distractors to unseen in-family instances and cross-family nuisance shifts? Third, do Prototype-Based Support Filtering and Sparse Control-Factor Selection contribute as intended? Finally, do these gains incur substantial training or planning costs?

4.1 Experimental Setup

Environments. We evaluate DSSF in four continuous-control environments: Two-Room, Reacher, Push-T, and OGBench-Cube, covering 2D navigation, planar robotic-arm control, contact-rich 2D manipulation, and 3D robotic manipulation, respectively. Each environment is evaluated with 0 (clean), 1, 3, or 5 randomly overlaid visual distractors; Figure 3 shows representative inputs. During training, DSSF augments clean observations with transparent toy-style assets sampled from a PrismLayersPro distractor bank. At evaluation, we use disjoint held-out assets from the same appearance family and two unseen distractor-appearance families, Shapes and Real-object. All families share the same overlay mechanism; Appendix A details their sources, construction procedures, and Static/Dynamic evaluation protocols.

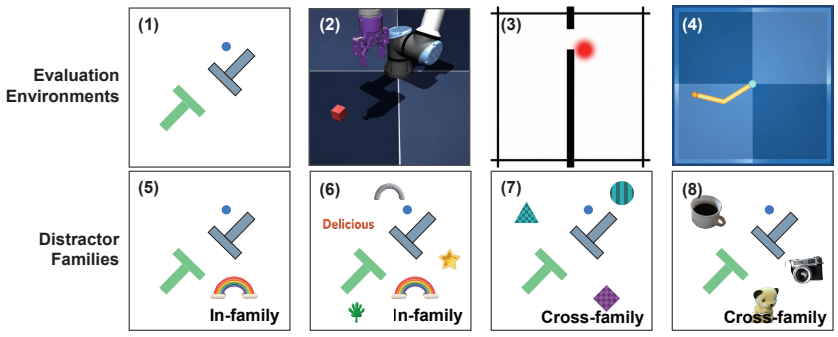


Figure 3: Evaluation environments and distractor families. Top: Push-T, OGBench-Cube, Two-Room, and Reacher. Bottom: in-family distractors with one and five instances, followed by cross-family geometric shapes and real-object images.

Baselines and Training Protocol. We compare DSSF with LeWM, PLDM, DINO-WM, and DINO-WM+Prop (Maes et al., 2026; Sobal et al., 2025; Zhou et al., 2025). DINO-WM uses a

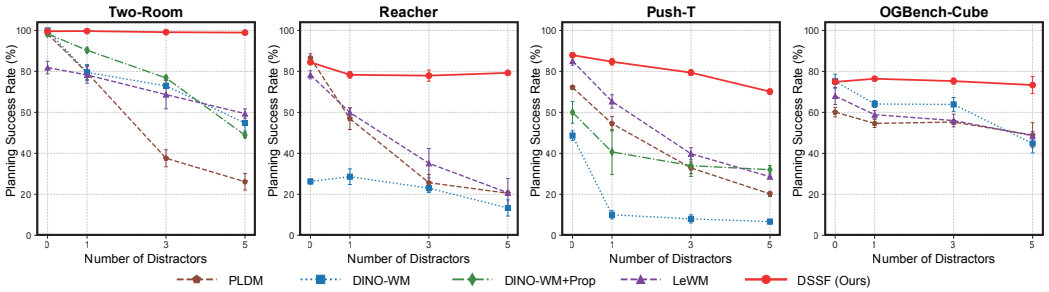


Figure 4: Robust planning under increasing visual distractors. We report the planning success rates of different methods in the four environments with 0, 1, 3, and 5 visual distractors. DSSF maintains more stable planning performance as the visual interference increases. Complete numerical results are provided in Appendix G.

frozen DINOv2 visual encoder, while DINO-WM+Prop additionally receives proprioceptive inputs. Following the task coverage in LeWM (Maes et al., 2026), we report DINO-WM+Prop results only for Two-Room and Push-T and mark the remaining tasks as unavailable. DSSF uses ResNet-18 as its visual backbone; Appendix C compares alternative backbones and motivates this choice. For controlled module-level ablations, CNN-LeWM denotes LeWM with its visual encoder replaced by the same ResNet-18 backbone, to which we add DSSF and/or the auxiliary action decoder. All methods use the same offline trajectories, number of training epochs, goal-conditioned planning protocol, and CEM budget. Each condition is evaluated over 500 episodes for each of three random seeds, and we report the mean $\pm$ standard deviation across seeds. Our primary metric is the planning success rate, denoted by $SR@n$ for $n \in \{0, 1, 3, 5\}$ distractors. Appendix D reports parameter counts, FLOPs, and end-to-end CEM latency, while Appendix E details the choices of the number of Support prototypes and the Control Top-K ratio.

4.2 Robust Planning under Increasing Visual Distractors

We first evaluate the planning robustness of different world models as visual interference increases. Specifically, we add 0, 1, 3, or 5 visual distractors to Two-Room, Reacher, Push-T, and OGBench-Cube and report the corresponding planning success rates in Fig. 4. Except in the clean setting, all distractors in this experiment are procedurally generated random geometric shapes that never appear during training. This experiment therefore also evaluates planning robustness under increasingly severe cross-family nuisance shifts. In the clean setting, DSSF performs comparably to the best-performing baseline in all four environments, indicating that its improved robustness does not come at the expense of standard planning performance. As the number of distractors increases, all baselines exhibit substantial performance degradation, whereas DSSF consistently shows the smallest decline across the four environments. From the clean setting to five distractors, its success rate decreases by only 0.67, 5.20, 17.80, and 1.54 percentage points, respectively. These results show that DSSF more reliably preserves latent-planning performance under increasing visual interference.

4.3 Zero-Shot Generalization to Unseen Distractor Appearances

To test whether DSSF learns transferable robustness rather than memorizing training distractors, we evaluate it on Push-T using unseen in-family instances and two cross-family distractor-appearance distributions: geometric shapes and real-object images. No fine-tuning or test-time adaptation is used. For each family, Static keeps distractor locations fixed between the initial and goal observations, whereas Dynamic changes them, making latent-space goal matching more challenging. For consistency, the Clean and Shapes-Static columns reuse the corresponding Push-T evaluation runs reported in Table 7. As shown in Table 1, DSSF achieves the highest success rate in all six nuisance settings. Under cross-family Shapes, it outperforms the strongest baseline by 38.13 and 11.81 percentage points in the Static and Dynamic settings, respectively; under Real-object distractors, the corresponding gains are 30.65 and 18.73 points. These results indicate that DSSF generalizes beyond the appearance family of its training distractors, including when distractor correspondence between the current and goal observations is disrupted.

Table 1: Zero-shot generalization across distractor appearances on Push-T. **In-family** uses unseen distractor instances from the same appearance family as training, while **Shapes** and **Real-object** introduce unseen cross-family distractors. In **Static**, distractor locations are shared between the initial and goal observations; in **Dynamic**, their locations differ. All non-clean conditions use five distractors per observation. No fine-tuning or test-time adaptation is used.

Method	Clean	In-family		Cross-family: Shapes		Cross-family: Real-object	
		Static ↑	Dynamic ↑	Static ↑	Dynamic ↑	Static ↑	Dynamic ↑
PLDM	72.20 ± 0.92	53.93 ± 2.34	50.87 ± 0.99	20.20 ± 1.31	13.13 ± 1.36	45.13 ± 0.83	40.27 ± 0.64
DINO-WM	48.67 ± 2.31	39.10 ± 2.13	34.95 ± 0.72	6.67 ± 1.15	6.69 ± 2.31	33.87 ± 0.34	27.11 ± 1.09
DINO-WM+Prop	60.00 ± 5.29	53.89 ± 1.25	47.71 ± 2.64	32.00 ± 2.00	21.12 ± 1.56	51.02 ± 2.78	41.80 ± 1.94
LeWM	85.07 ± 2.04	59.93 ± 1.17	53.53 ± 2.30	28.60 ± 0.20	12.60 ± 0.35	39.40 ± 1.25	30.60 ± 1.11
DSSF (Ours)	$\mathbf{87.93 \pm 0.99}$	$\mathbf{85.53 \pm 2.14}$	$\mathbf{55.20 \pm 2.00}$	$\mathbf{70.13 \pm 1.21}$	$\mathbf{32.93 \pm 0.23}$	$\mathbf{81.67 \pm 0.50}$	$\mathbf{60.53 \pm 2.01}$

4.4 Qualitative Analysis of DSSF

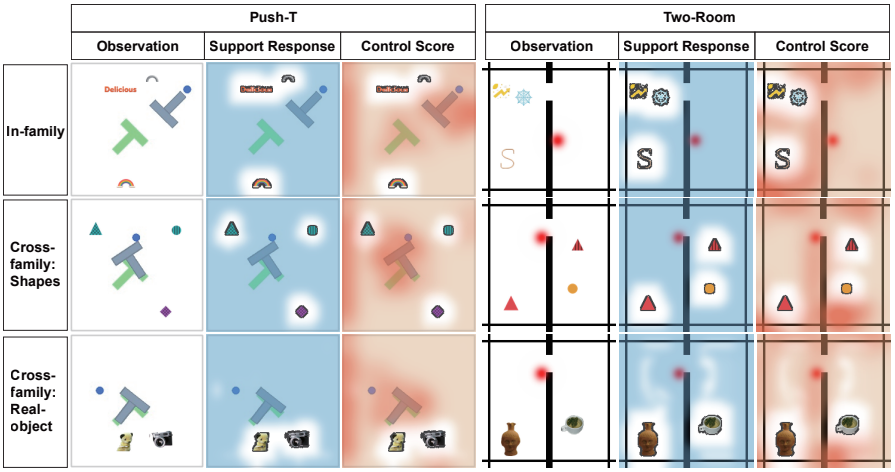


Figure 5: Qualitative visualization of DSSF on Push-T and Two-Room. Columns show the observation, prototype-based Support response, and Control score; rows show in-family, cross-family Shapes, and cross-family Real-object distractors.

As shown in Fig. 5, Support Filtering suppresses distractor regions while preserving the principal task structure across nuisance families, including Shapes and Real-object appearances unseen during training. The Control score further concentrates on task-relevant regions, primarily the agent and T-shaped block in Push-T and the agent and navigation structure in Two-Room. Together, these examples support the intended division of labor: Support Filtering rejects unsupported nuisance factors, while Control-Factor Selection retains information relevant to latent dynamics. Appendix B quantifies this behavior by treating $1 - \bar{R}_{t,i}$ as a local distractor score. Across the four tasks, Stage I achieves macro-average AUROC values of 99.97%, 97.74%, and 99.77% on held-out in-family, Shapes, and Real-object distractors, respectively. These results indicate that its filtering is not limited to memorizing the distractor instances used for optimization.

4.5 Ablation Studies

To analyze the contribution of each component to robustness against visual distractors, we conduct two sets of ablation studies. Except for the component being removed, all variants use the same training data, distractor augmentation, visual backbone, latent predictor, and planning protocol to ensure a fair comparison.

Module-level ablation. Using CNN-LeWM—LeWM with its visual encoder replaced by ResNet-18—as the common base, we evaluate the 2×2 combinations of DSSF and auxiliary action decoding: neither component, each component alone, and both components (Full model). As shown in Table 2, DSSF alone reduces $\Delta_{0 \to 5}$ from 65.60 to 28.93 percentage points, but lowers the clean success

Table 2: Module-level ablation on Push-T under increasing numbers of visual distractors. All variants use the same distractor augmentation and planning protocol. $\Delta_{0\to5}$ denotes the decrease in success rate from the clean setting to five distractors; lower is better.

Method	SR@0 ↑	SR@1 ↑	SR@3 ↑	SR@5 ↑	$\Delta_{0\to5}$ ↓
CNN-LeWM	**88.93 ± 0.61**	63.87 ± 2.84	35.27 ± 2.55	23.33 ± 3.45	65.60
DSSF only	61.20 ± 1.25	55.33 ± 0.42	52.20 ± 0.35	32.27 ± 0.76	28.93
Action decode only	88.67 ± 0.83	63.80 ± 1.97	57.60 ± 0.72	37.20 ± 2.60	51.47
Full model	87.93 ± 0.99	**84.73 ± 1.63**	**79.40 ± 1.40**	**70.13 ± 1.21**	**17.80**

Table 3: Internal ablation study of DSSF on Push-T under structured distractors and Gaussian noise. All results report planning success rate (%). For Gaussian noise, Low, Medium, and High correspond to $\sigma = 0.03, 0.08$, and 0.15, respectively, with pixel values normalized to $[0, 1]$.

Method	**Number of Distractors**				**Gaussian Noise**		
	SR@0 ↑	SR@1 ↑	SR@3 ↑	SR@5 ↑	Low	Medium	High
Support Only	86.20 ± 0.35	81.27 ± 1.63	75.47 ± 0.61	**70.13 ± 0.61**	36.00 ± 5.29	18.00 ± 2.00	10.00 ± 5.29
Control Only	78.47 ± 1.14	76.20 ± 0.80	69.60 ± 0.40	63.60 ± 1.04	**80.67 ± 1.15**	**66.00 ± 9.17**	**35.33 ± 3.06**
Full DSSF	**87.93 ± 0.99**	**84.73 ± 1.63**	**79.40 ± 1.40**	**70.13 ± 1.21**	80.00 ± 3.46	64.67 ± 6.11	32.67 ± 7.57

rate to 61.20%, suggesting that factor selection without action-recovery supervision may discard dynamics-relevant information. Action decoding alone preserves clean performance (88.67%) but still incurs a 51.47-point drop under five distractors. Combining both components retains a clean success rate of 87.93% and achieves 70.13% with five distractors, yielding the smallest performance drop of 17.80 points. These results indicate that the two components are complementary: DSSF filters factors lacking training-distribution support, while action decoding encourages the retained representation to capture state-transition information. Appendix F provides qualitative evidence for this effect, showing that action decoding shifts Control-factor selection from static backgrounds toward the pusher and T-shaped block.

Internal ablation of DSSF. We compare three variants: Support Only, which retains Prototype-Based Support Filtering and densely aggregates the features passing the Support Gate; Control Only, which fixes the Support Gate to one and applies only Sparse Control-Factor Selection; and Full DSSF, which combines both stages. We evaluate them under local structured distractors of increasing number and global Gaussian noise. As shown in Table 3, Support Only consistently outperforms Control Only under structured distractors, while Full DSSF improves over Support Only with zero to three distractors and matches it with five. This suggests that Support Filtering plays the primary role in suppressing localized nuisance factors, with Control Selection providing additional removal of control-irrelevant information. Under Gaussian noise, however, Support Only degrades sharply, whereas the two variants containing Control Selection remain substantially more robust. Although Control Only has slightly higher mean success rates than Full DSSF, the differences are small relative to the reported variability. Overall, Support Filtering is most effective for localized structured nuisances, while Control Selection is important for robustness to global perturbations.

5 Conclusion

The central insight of this work is that the robustness of visual world models depends not only on the predictive quality of latent dynamics, but also on which visual information is allowed to enter the latent state. Our experiments show that prototype-based support filtering is particularly effective at suppressing local structured distractors, whereas sparse control-factor selection is more robust to global visual noise. Although our evaluation covers distractor instances and appearance categories unseen during training, all structured distractors are introduced through the same image-overlay mechanism. Therefore, the current results do not yet establish that DSSF can handle other forms of visual shift, such as changes in illumination, background, viewpoint, or camera configuration. Future work will evaluate DSSF under visual variations generated by different mechanisms and on real-robot observations, while exploring adaptive support estimation and factor-selection strategies.

AI USE STATEMENT

We used OpenAI's ChatGPT/Codex to assist with translation and software implementation. We additionally used these tools to edit portions of the manuscript for clarity and concision, locate and summarize relevant literature, format LaTeX and references, and check the numerical and cross-reference consistency of the manuscript. The research method and experimental protocol were determined by the authors. Generative AI was not used to generate, select, or process experimental data, and all reported measurements were obtained from experiments conducted by the authors. All AI-assisted text and code were manually reviewed, revised, and verified by the authors. Citations were checked against the original sources, and AI-assisted code was validated through author-run experiments and tests. The authors take full responsibility for the final manuscript, code, claims, and results.

REPRODUCIBILITY STATEMENT

To support reproducibility, the supplementary material provides the constructed nuisance datasets and our implementations of the evaluated baselines. Section 4 specifies the evaluation protocol, metrics, and random-seed setting. Appendix A documents the distractor sources, generation procedures, and test-set construction; Appendix B reports the quantitative Stage-I analysis; Appendix E explains the selection of the main DSSF hyperparameters; and Appendices G and H provide numerical results and training configurations. These details are intended to support independent reimplementation and verification. The core DSSF implementation is undergoing internal team review and will be released in stages as the corresponding components complete that review.

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*, pp. 15619–15629, 2023. doi: 10.1109/CVPR52729.2023.01499. URL https://doi.org/10.1109/CVPR52729.2023.01499.

Junwen Chen, Heyang Jiang, Yanbin Wang, Keming Wu, Ji Li, Chao Zhang, Keiji Yanai, Dong Chen, and Yuhui Yuan. PrismLayers: Open data for high-quality multi-layer transparent image generative models. *arXiv preprint arXiv:2505.22523*, 2025. doi: 10.48550/arXiv.2505.22523. URL https://arxiv.org/abs/2505.22523.

Jeongsoo Ha, Kyungsoo Kim, and Yusung Kim. Dream to generalize: Zero-shot model-based reinforcement learning for unseen visual distractions. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(6):7802–7810, 2023. doi: 10.1609/aaai.v37i6.25945. URL https://doi.org/10.1609/aaai.v37i6.25945.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2555–2565. PMLR, 2019. URL https://proceedings.mlr.press/v97/hafner19a.html.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640:647–653, 2025. doi: 10.1038/s41586-025-08744-2. URL https://doi.org/10.1038/s41586-025-08744-2.

HoKyun Im, Euijin Jeong, Andrey Kolobov, Jianlong Fu, and Youngwoon Lee. TwinVLA: Data-efficient bimanual manipulation with twin single-arm vision-language-action models. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/65475a59e2a2cb707c9462e8a308e980-Abstract-Conference.html.

Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, and Vittorio

Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *International Journal of Computer Vision*, 128(7):1956–1981, 2020. doi: 10.1007/s11263-020-01316-z. URL https://doi.org/10.1007/s11263-020-01316-z.

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

NVIDIA et al. Cosmos world foundation model platform for physical ai. *arXiv preprint arXiv:2501.03575*, 2025. URL https://arxiv.org/abs/2501.03575.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588:604–609, 2020. doi: 10.1038/s41586-020-03051-4. URL https://doi.org/10.1038/s41586-020-03051-4.

Uladzislau Sobal, Wancong Zhang, Kyunghyun Cho, Randall Balestriero, Tim G. J. Rudner, and Yann LeCun. Learning from reward-free offline data: A case for planning with latent dynamics models. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1465. URL https://doi.org/10.52202/085713-1465.

Ruixiang Sun, Hongyu Zang, Xin Li, and Riashat Islam. Learning latent dynamic robust representations for world models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 47234–47260. PMLR, 2024. URL https://proceedings.mlr.press/v235/sun24n.html.

Leonardo F. Toso, Davit Shadunts, Yunyang Lu, Nihal Sharma, Donglin Zhan, Nam H. Nguyen, and James Anderson. Learning invariant visual representations for planning with joint-embedding predictive world models, 2026. URL https://arxiv.org/abs/2602.18639.

Yuqi Wang, Xinghang Li, Wenxuan Wang, Junbo Zhang, Yingyan Li, Yuntao Chen, Xinlong Wang, and Zhaoxiang Zhang. Unified vision-language-action model. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/82f625a28d822d2748b9f5c4f9a89bb9-Abstract-Conference.html.

Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=GY6-6sTvGaf.

Zhenghao Zhang, Yuanxiang Wang, Zhenyu Guan, Yujia Yang, Bingkang Shi, Tianyu Zong, Hongzhu Yi, Guoqing Chao, Xingchen Chen, Tiankun Yang, Chenxi Bao, Tao Yu, Jingjing Zhou, and Jungang Xu. Delta-JEPA: Learning action-sensitive world models via latent difference decoding, 2026. URL https://arxiv.org/abs/2606.31232.

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

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

A Visual Distractor Sets and Evaluation Protocol

This section describes the three visual distractor families used for zero-shot evaluation and the common evaluation protocol. The distractors modify only the RGB inputs of the current observation and goal image, without changing the underlying physical state or environment dynamics.

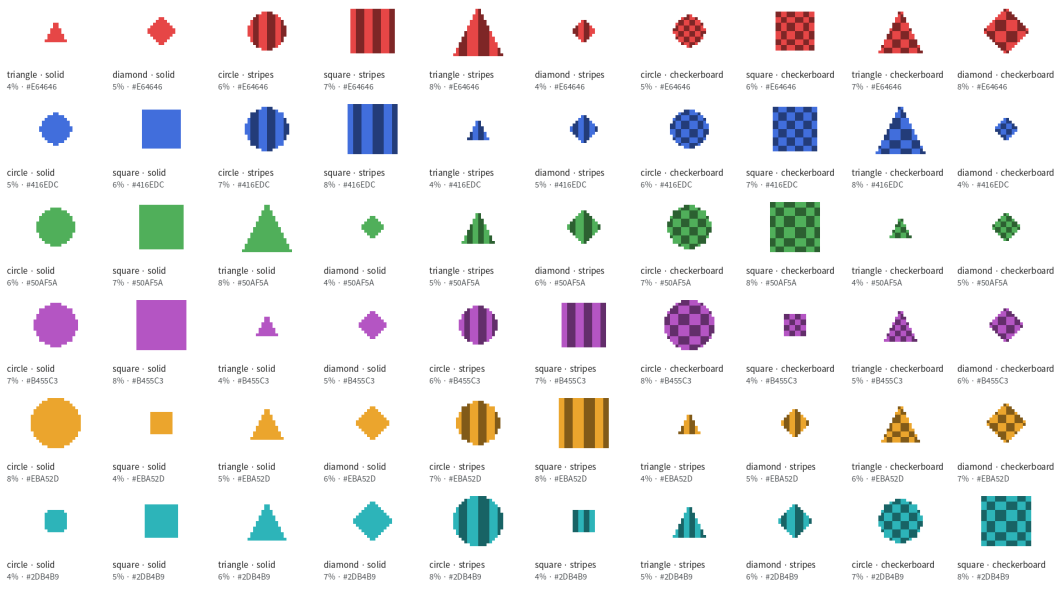
A.1 Distractor Families



(a) In-family



(b) Real-object



(c) Shapes

Figure 6: Examples of the three distractor families: (a) held-out toy-style assets from PrismLayersPro (In-family), (b) object cutouts from Open Images (Real-object), and (c) procedurally generated geometric patterns (Shapes). Assets are enlarged for visibility; their displayed sizes do not reflect evaluation-time sizes.

Figure 6 shows representative appearances from the three distractor families. In-family distractors evaluate instance-level generalization within the training appearance family, whereas Shapes and Real-object distractors evaluate cross-family generalization.

In-family. The training distractor bank consists of transparent *toy-style* image assets from PrismLayersPro (Chen et al., 2025). Evaluation uses held-out assets from the same appearance family that

are never used for parameter updates; each evaluation instance samples one object from each of five distinct source-asset groups.

Cross-family: Shapes. This set combines four shapes (circle, square, triangle, and diamond), three textures (solid, stripes, and checkerboard), and six colors, yielding $4 \times 3 \times 6 = 72$ discrete appearances. The generation and scaling procedure is given in Algorithm 1 and is independent of the image-asset bank used during training.

Cross-family: Real-object. We use object cutouts from the Open Images validation split (Kuznetsova et al., 2020), comprising five instances from each of 20 categories. Each evaluation instance samples one object from each of five distinct categories. These images are excluded from training and are used to assess cross-family generalization to realistic object appearances.

All three distractor families are introduced by resizing and compositing localized visual elements at sampled image positions. The protocol therefore evaluates generalization across distractor appearances, counts, and current–goal layouts while holding the nuisance-generation mechanism fixed.

Algorithm 1: Procedural shape generation

Input: Number of distractors $N = 5$; image size $H \times W$.

for $j = 1, \ldots, N$ **do**
  Sample a shape $g \in \{\text{circle, square, triangle, diamond}\}$.
  Sample a texture $r \in \{\text{solid, stripes, checkerboard}\}$.
  Sample a color c from the six-color palette.
  Sample target area ratio $u \sim \mathcal{U}(0.002, 0.006)$.
  Render an RGBA sprite D_j using (g, r, c).
  Resize D_j to target nontransparent area uHW, preserving its aspect ratio.
  Limit its longest side to $0.28 \min(H, W)$.
end for
return $\{D_j\}_{j=1}^{N}$.

In-family and Real-object distractors use image assets with the same size-sampling rule.

A.2 Evaluation Settings

Count and size. We use a common count and size configuration for all three distractor families. Each image contains between zero and five injected objects, and the target nontransparent area of each object is sampled uniformly from 0.2%–0.6% of the image area. For a 224×224 input, this range corresponds to approximately 100–301 pixels. Each object is resized while preserving its aspect ratio, and the longest side of its bounding box is capped at 28% of the shorter image dimension (approximately 63 pixels). Consequently, the actual area may be constrained by this side-length limit, and objects do not share a fixed width or height. Figure 7 illustrates Static injection on Push-T, Two-Room, Reacher, and OGBench-Cube. Within each row, the three distractor families are added to the same clean observation to facilitate comparison of their appearance and scale in the task image. The figure illustrates input construction only and does not report additional planning results.

Static versus Dynamic. In the **Static** setting, distractors occupy the same locations in the current observation and goal image. In the **Dynamic** setting, object identities, appearances, and sizes remain fixed, but their positions in the goal image are resampled. Subsequent online observations retain the layout of the initial observation. Dynamic evaluation therefore measures spatial misalignment between the current and goal observations rather than distractors that move continuously over time. All methods are evaluated zero-shot, without fine-tuning or test-time adaptation. For a given task and random seed, all methods share the same physical evaluation instances and use identical distractor layouts under each nuisance condition to ensure a fair comparison.

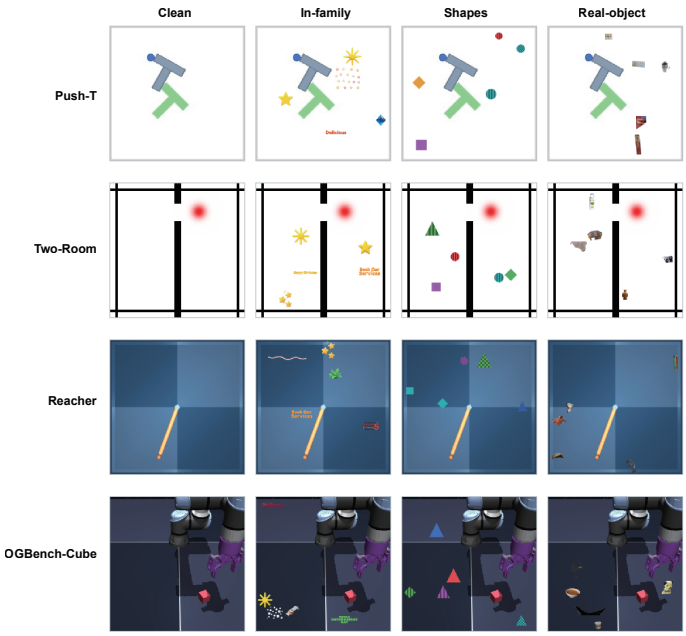


Figure 7: Static distractor examples on four tasks. Each row shows the same clean observation followed by three independently sampled layouts, each containing five distractors from one family. These examples illustrate the visual inputs, not additional planning results or the goal-position changes used in Dynamic evaluation.

B Quantitative Analysis of Stage-I Support Filtering

We quantitatively evaluate whether the Stage-I Support Response separates distractor-affected local regions from valid clean regions beyond the distractor assets used for gradient-based optimization. This analysis treats Stage I as a local binary separator; it does not interpret the Support Response as a calibrated probability, density estimate, or general OOD score.

Protocol. For each of Push-T, Two-Room, Reacher, and OGBench-Cube, we evaluate one trained DSSF checkpoint on 500 deterministically sampled cases from the validation trajectory split, with one frame selected per case. Each image contains five distractors. We consider three test families: toy-style assets held out from gradient-based optimization, cross-family geometric Shapes, and cross-family Real-object cutouts. For a given case, the three families share the same clean frame, object centers, and target alpha areas; the target area of each object is sampled from 0.2%–0.6% of the image area. This controls for placement and nominal size when comparing appearance families.

We evaluate the 28×28 Stage-I grid. A cell containing visible distractor pixels is labeled positive, and the nuisance score is defined as $1 - \bar{R}_{t,i}$, where $\bar{R}_{t,i}$ is the normalized Support Response. A higher nuisance score therefore indicates lower support. Cells directly covered by a distractor are retained as positives. Valid clean cells serve as negatives, while clean cells within three grid cells of a distractor are excluded to avoid ambiguous receptive-field contamination, following the Support-loss construction used during training.

Metrics. We report AUROC, AUPRC, and FPR@95TPR. The latter is the fraction of valid clean cells incorrectly rejected at a per-case threshold that attains at least 95% nuisance recall. Metrics are computed separately for each case and then averaged. The 95% confidence intervals are obtained from 2,000 bootstrap resamples of the 500 cases, rather than by treating spatial cells within an image as independent samples. The four-task macro average assigns equal weight to each task.

Results. As shown in Table 4, Stage I nearly perfectly separates held-out in-family assets, with macro-average AUROC/AUPRC of $99.97\%/99.96\%$ and FPR@95TPR of 0.35%. Separation also transfers to the two cross-family appearance sets: the macro-average AUROC/AUPRC is

Table 4: Stage-I separation of distractor-affected and valid clean grid cells across appearance families. Values are percentages and report the case-macro mean with the 95% case-bootstrap confidence interval in brackets. Higher AUROC and AUPRC are better; lower FPR@95TPR is better.

Task	Distractor family	AUROC ↑	AUPRC ↑	FPR@95TPR ↓
Push-T	In-family held-out	99.87 [99.80, 99.93]	99.86 [99.78, 99.92]	1.40 [0.58, 2.39]
	Cross-family: Shapes	90.99 [90.08, 91.88]	89.48 [88.41, 90.52]	53.86 [49.43, 58.10]
	Cross-family: Real-object	99.96 [99.94, 99.98]	99.91 [99.86, 99.94]	0.01 [0.00, 0.02]
Two-Room	In-family held-out	100.00 [100.00, 100.00]	100.00 [100.00, 100.00]	0.00 [0.00, 0.00]
	Cross-family: Shapes	100.00 [100.00, 100.00]	100.00 [100.00, 100.00]	0.00 [0.00, 0.00]
	Cross-family: Real-object	100.00 [100.00, 100.00]	100.00 [100.00, 100.00]	0.00 [0.00, 0.00]
Reacher	In-family held-out	100.00 [100.00, 100.00]	100.00 [100.00, 100.00]	0.00 [0.00, 0.00]
	Cross-family: Shapes	100.00 [100.00, 100.00]	100.00 [100.00, 100.00]	0.00 [0.00, 0.00]
	Cross-family: Real-object	100.00 [100.00, 100.00]	100.00 [100.00, 100.00]	0.00 [0.00, 0.00]
OGBench-Cube	In-family held-out	100.00 [100.00, 100.00]	100.00 [100.00, 100.00]	0.00 [0.00, 0.00]
	Cross-family: Shapes	99.98 [99.96, 99.99]	99.94 [99.91, 99.97]	0.01 [0.00, 0.01]
	Cross-family: Real-object	99.11 [98.87, 99.32]	98.41 [98.01, 98.80]	9.30 [6.94, 12.04]
4-task macro	In-family held-out	99.97 [99.95, 99.98]	99.96 [99.95, 99.98]	0.35 [0.13, 0.60]
	Cross-family: Shapes	97.74 [97.51, 97.96]	97.36 [97.10, 97.61]	13.47 [12.43, 14.50]
	Cross-family: Real-object	99.77 [99.71, 99.82]	99.58 [99.48, 99.68]	2.33 [1.69, 2.98]

$97.74\%/97.36\%$ for Shapes and $99.77\%/99.58\%$ for Real-object distractors. These results indicate that Stage I does not rely only on memorizing the asset instances used for gradient-based optimization.

The transfer is nevertheless task dependent. Push-T with Shapes is the principal failure mode, with AUROC/AUPRC of $90.99\%/89.48\%$ and FPR@95TPR of 53.86%. At the deployed threshold $\bar{R} = 0.5$, Stage I rejects 87.17% of nuisance cells while incorrectly rejecting 0.95% of valid clean cells; reaching 95% nuisance recall therefore requires moving the threshold into a dense region of clean responses. A plausible explanation is that simple geometric distractors share local shape and color cues with the normal Push-T elements. OGBench-Cube with Real-object distractors is the second most difficult condition, with FPR@95TPR of 9.30%, although its AUROC and AUPRC remain above 98%.

Scope and limitations. The held-out in-family assets are excluded from gradient-based optimization but are used for validation forward passes, so this condition measures transfer beyond the optimization assets rather than exposure-free evaluation. In addition, all three families use the same resizing, positioning, and alpha-compositing mechanism. The results therefore establish transfer of supervised local separation across distractor instances and most cross-family appearances, but not calibrated density estimation or general OOD detection. Finally, each task is evaluated with one trained checkpoint; the confidence intervals quantify case-level uncertainty and do not include variation across independently trained models.

C Encoder Architecture Ablation

To select a backbone that balances robustness to visual distractors and computational cost, Table 5 compares four encoders under 0, 1, 3, and 5 distractors and reports their parameter counts and per-image forward-pass FLOPs. All rows use the LeWM predictor and training objective and differ only in their visual encoder; the ResNet-18 configuration is denoted CNN-LeWM in the module-level ablation. Each encoder uses the same three-seed protocol with 500 evaluation trajectories per seed, as described in the main paper. ResNet-18 achieves a clean success rate of 88.93%, close to the best-performing DenseNet-121 (90.47%), while obtaining the highest mean success rate in all three distractor settings. With five distractors, it outperforms the second-best ConvNeXt-Tiny by 9.06 percentage points while using 59.8% fewer parameters and 59.3% fewer FLOPs. ResNet-18 also requires fewer FLOPs than DenseNet-121. Although EfficientNet-B0 is the most lightweight encoder, its success rates under distractors are substantially lower. We therefore select ResNet-18 as the visual backbone based on its balance between distractor robustness and computational cost.

Table 5: Backbone comparison under increasing visual distractors. Success rates are reported as mean ± standard deviation (%). Params and FLOPs cover the encoder only, with FLOPs measured per image. The highest mean success rates and lowest costs are bold.

Encoder	SR@0 ↑	SR@1 ↑	SR@3 ↑	SR@5 ↑	Params (M) ↓	FLOPs (G) ↓
ConvNeXt-Tiny	88.40 ± 2.31	63.07 ± 2.39	28.67 ± 1.47	14.27 ± 0.90	27.820	8.910
DenseNet-121	$\mathbf{90.47} \pm 1.29$	57.87 ± 1.70	22.60 ± 2.20	11.67 ± 0.64	6.954	5.666
EfficientNet-B0	84.47 ± 1.75	40.13 ± 1.14	10.87 ± 0.58	6.47 ± 1.67	**4.008**	**0.769**
ResNet-18	88.93 ± 0.61	$\mathbf{63.87} \pm 2.84$	$\mathbf{35.27} \pm 2.55$	$\mathbf{23.33} \pm 3.45$	11.177	3.627

D Computational Efficiency

We further compare the computational efficiency of the different methods. All experiments are conducted on a single NVIDIA L40S GPU with 48 GB of memory, on a server equipped with an Intel(R) Xeon(R) Platinum 8480+ processor and 24 CPU cores. For a fair comparison, all methods use a consistent input specification and CEM planning configuration. Table 6 reports the number of model parameters, the FLOPs of a single forward step, and the latency of a complete CEM planning procedure. The FLOPs include visual encoding and one latent-prediction step, whereas Planning Time measures the end-to-end latency of the full CEM optimization. DSSF uses fewer parameters and fewer single-step FLOPs than its LeWM baseline while retaining low planning latency. Although Support Filtering and Control Selection introduce additional encoding overhead, the complete CEM planning procedure of DSSF remains below one second and is substantially faster than the DINOv2-based DINO-WM variants.

Table 6: Computational efficiency comparison. All results are measured on a single NVIDIA L40S GPU (48 GB), with 24 CPU cores and an Intel(R) Xeon(R) Platinum 8480+ processor. Planning Time is the end-to-end latency, in milliseconds, of one complete CEM planning procedure.

Method	Params (M) ↓	FLOPs (G) ↓	Planning Time (ms) ↓
PLDM	18.03	10.26	482.60 ± 0.77
DINO-WM	41.75	80.98	76806.79 ± 164.19
DINO-WM+Prop	42.25	81.74	75146.53 ± 181.48
LeWM	18.03	10.26	326.24 ± 0.28
DSSF (Ours)	**13.44**	**6.41**	442.08 ± 0.36

E Selection of Support Prototypes and Control Top-K

Number of support prototypes. We determine the number of Support prototypes K_s based on qualitative observations during experimentation. For $K_s \in \{20, 30, 40, 50\}$, the model consistently distinguishes non-distractor regions from distractor regions across all four tasks: content compatible with the clean training data, including normal backgrounds, receives high Support responses, whereas responses in distractor regions are suppressed. Figure 8 shows representative examples of this filtering behavior. We therefore choose the smallest empirically effective setting, $K_s = 20$, to reduce prototype storage and matching costs while preserving this discriminative behavior. This is a visualization-based empirical choice rather than an exhaustive search for the optimal number of prototypes.

Control Top-K budget. We use foreground-grid occupancy in clean images as a reference for the spatial budget K_c of Control factors. A 224×224 image is partitioned into a 28×28 grid of 784 cells, each corresponding to an 8×8 pixel region. A cell is marked as occupied if it contains any non-background pixel, without further dilation. Figure 9 shows representative examples from the four tasks. For each task, we measure 500 clean initial evaluation observations under each of three random seeds, yielding 1,500 observations in total. The mean occupancy ratios for Push-T, Two-Room, Reacher, and OGBench-Cube are 29.91%, 30.67%, 3.25%, and 31.62%, respectively.

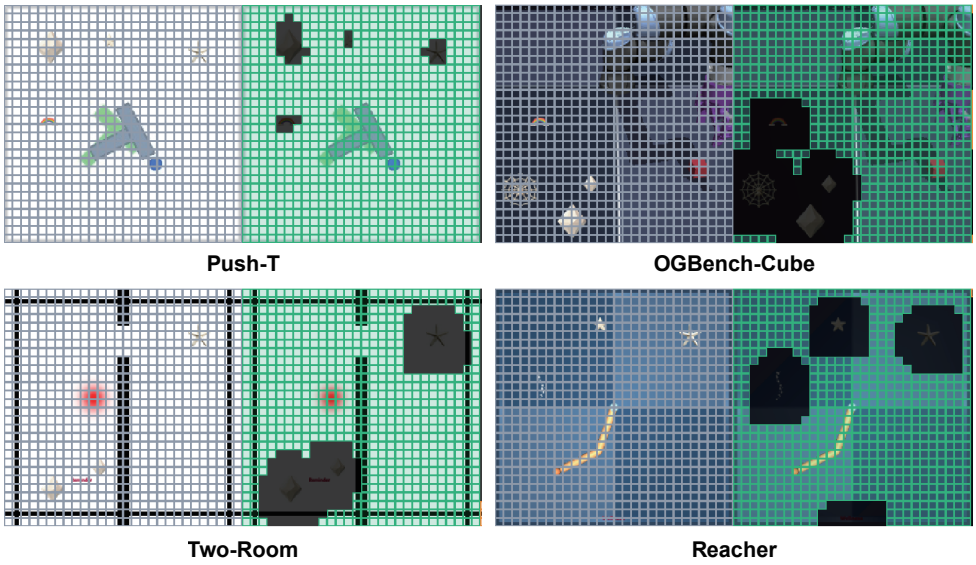


Figure 8: Representative examples of prototype-based support filtering on four tasks. Each pair shows the input on the left and retained grid positions in green on the right. Non-distractor regions, including normal backgrounds, are largely retained, while distractor regions are suppressed.

Because the three tasks with higher occupancy exhibit a typical spatial scale of approximately 30%, we adopt a shared 30% selection budget:

$$K_c = \lfloor 0.30 \times 784 \rfloor = 235. \tag{6}$$

This ratio provides an empirical basis for sparse selection; it is neither an upper bound on per-frame occupancy nor a guarantee that all foreground content is retained. The occupancy statistics include scene structures such as walls and goal markers and are used only to set the selection budget. The actual Control factors are selected from candidate locations passing the Support Gate according to their Control scores, rather than being specified by foreground masks.

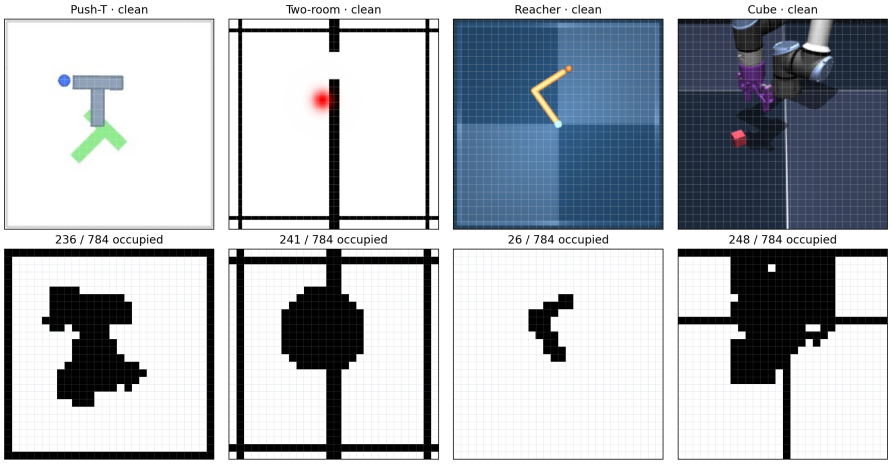
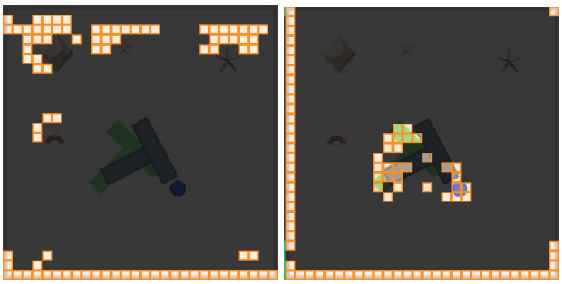


Figure 9: Foreground occupancy on the 28×28 factor grid. Top: clean observations; bottom: occupied cells in black. Each example is closest to its task's median occupancy among 1500 clean initial observations. The occupied-cell counts are 236, 241, 26, and 248 out of 784. These are input-image statistics, not predicted Support Gates or selected Control factors.



(a) Without action decoding (b) With action decoding

Figure 10: Qualitative effect of auxiliary action decoding on Control-factor selection in Push-T. Orange cells denote selected locations. Action decoding shifts selection from the background and image boundaries toward the pusher and T-shaped block. The grids have different resolutions and are not directly comparable quantitatively.

F Effect of Auxiliary Action Decoding

Figure 10 compares Control-factor selection on Push-T with and without auxiliary action decoding. Without this objective, the selected cells concentrate largely on the static background and image boundaries. With action decoding, the selection shifts toward the pusher and the T-shaped block, although some boundary cells remain selected. The no-decoding variant removes only the auxiliary action-decoding loss ($\lambda_{\text{act}} = 0$); actions remain inputs to the dynamics predictor.

This behavior highlights why Support Filtering alone is insufficient for control-relevant state construction. Normal backgrounds are also supported by the training distribution and can therefore pass the Support Gate despite being unrelated to task dynamics. By requiring actions to be recovered from adjacent latent states, the auxiliary objective encourages the representation to retain factors that reflect action-induced transitions. The visualization complements the planning improvements in Table 2, providing qualitative evidence that action decoding guides the Control-Factor Selector toward dynamics-relevant regions.

G Detailed Planning Results

Table 7 provides the complete numerical results corresponding to Fig. 4 in the main paper. $SR@n$ denotes the planning success rate with n distractors, with $SR@0$ corresponding to the clean setting. Results are reported as percentages (mean $\pm$ standard deviation). DINO-WM+Prop denotes the variant with additional proprioceptive inputs.

H Training Details

Training setup. We follow the data-sampling and dynamics-prediction settings of LeWM (Maes et al., 2026), replacing its visual state-construction module with DSSF. The main hyperparameters are summarized in Table 8. Training clips use a frame skip of 5, and the five raw actions between adjacent observations are grouped into a single action block, denoted by a_t in our formulation. The predictor receives action conditioning through AdaLN and uses causal attention for next-state prediction. The current and next states are produced by a shared encoder; gradients are not stopped through the target representation, and no EMA target encoder is used. The choices of K_s and K_c are discussed in Appendix E.

Support response and supervision. The raw response $R_{t,i}$ defined in Eq. 2 is the maximum cosine similarity and therefore lies in $[-1, 1]$. Before thresholding and BCE supervision, we linearly map it to $[0, 1]$:

$$\bar{R}_{t,i} = \frac{R_{t,i} + 1}{2}. \tag{7}$$

Table 7: Detailed planning success rates (%) corresponding to Fig. 4. Values are mean ± standard deviation. The highest mean in each task and column is bold; — denotes results not reported.

Task	Method	SR@0 ↑	SR@1 ↑	SR@3 ↑	SR@5 ↑
Push-T	PLDM	72.20 ± 0.92	54.53 ± 3.52	32.87 ± 2.90	20.20 ± 1.31
	DINO-WM	48.67 ± 2.31	10.00 ± 2.00	8.00 ± 2.00	6.67 ± 1.15
	DINO-WM+Prop	60.00 ± 5.29	40.67 ± 11.02	34.00 ± 5.29	32.00 ± 2.00
	LeWM	85.07 ± 2.04	65.33 ± 3.26	39.80 ± 2.95	28.60 ± 0.20
	DSSF	$\mathbf{87.93} \pm 0.99$	$\mathbf{84.73} \pm 1.63$	$\mathbf{79.40} \pm 1.40$	$\mathbf{70.13} \pm 1.21$
Two-Room	PLDM	98.33 ± 1.15	79.40 ± 3.49	37.60 ± 4.16	26.13 ± 4.03
	DINO-WM	$\mathbf{100.00} \pm 0.00$	79.53 ± 3.87	72.93 ± 1.30	54.80 ± 3.94
	DINO-WM+Prop	98.33 ± 0.31	90.33 ± 1.15	76.80 ± 1.11	48.87 ± 1.15
	LeWM	81.87 ± 3.06	78.27 ± 4.00	68.67 ± 6.93	59.40 ± 2.31
	DSSF	99.60 ± 0.20	$\mathbf{99.67} \pm 0.23$	$\mathbf{99.13} \pm 0.23$	$\mathbf{98.93} \pm 0.23$
Reacher	PLDM	$\mathbf{86.60} \pm 2.00$	56.93 ± 5.28	25.67 ± 4.00	20.53 ± 1.15
	DINO-WM	26.27 ± 1.27	28.60 ± 3.83	23.00 ± 2.12	13.33 ± 3.95
	DINO-WM+Prop	—	—	—	—
	LeWM	78.33 ± 2.00	59.93 ± 2.31	35.13 ± 7.21	20.73 ± 7.02
	DSSF	84.47 ± 1.10	$\mathbf{78.33} \pm 1.63$	$\mathbf{77.93} \pm 2.73$	$\mathbf{79.27} \pm 0.81$
OGBench-Cube	PLDM	60.13 ± 2.31	54.60 ± 2.00	55.27 ± 1.15	48.93 ± 6.00
	DINO-WM	$\mathbf{75.20} \pm 3.42$	64.07 ± 1.68	63.87 ± 3.37	44.80 ± 4.57
	DINO-WM+Prop	—	—	—	—
	LeWM	68.00 ± 4.16	58.87 ± 2.00	56.00 ± 3.08	48.60 ± 2.00
	DSSF	74.87 ± 0.31	$\mathbf{76.40} \pm 0.80$	$\mathbf{75.27} \pm 1.51$	$\mathbf{73.33} \pm 4.16$

Table 8: Main training settings. Data sampling and predictor settings follow LeWM; the remaining entries specify the DSSF configuration used in our experiments.

Setting	Value
Data and predictor settings shared with LeWM	
Input resolution	224×224
Batch size	128
Frames per training clip	4
Frame-skip / actions per block	5
Latent dimension	192
Predictor layers / attention heads	6 / 16
Predictor dropout	0.1
SIGReg random projections	1024
DSSF configuration	
Training epochs	10
Learning rate	5×10^{-5}
Visual backbone	ResNet-18
Spatial factor grid	$28 \times 28 = 784$
Support prototypes K_s	20
Control selection budget K_c	235
Support threshold τ	0.5

Accordingly, the Support Gate threshold $\tau = 0.5$ is applied to $\bar{R}_{t,i}$, and the BCE loss also operates on this normalized response rather than on the raw cosine similarity. For a label $y \in \{0, 1\}$,

$$\mathrm{BCE}(\bar{R}, y) = -y \log \bar{R} - (1 - y) \log(1 - \bar{R}). \tag{8}$$

Here, $\bar{R}$ denotes the normalized support score. During training, grid cells directly covered by distractors serve as negative samples, while cells outside a dilation radius of $r = 3$ grid cells serve as positive samples; the intermediate neighborhood is excluded from the Support loss.

Discrete selection and surrogate gradients. We use a hard Support Gate and hard Top-K_c selection during both training and inference, ensuring identical forward selection rules. Because thresholding and ranking are non-differentiable, we use a Straight-Through Estimator (STE) during training. Let m^{hard} denote the binary selection mask and m^{soft} its continuous surrogate. We define

$$m^{\mathrm{STE}} = m^{\mathrm{soft}} + \mathrm{sg}\big(m^{\mathrm{hard}} - m^{\mathrm{soft}}\big), \tag{9}$$

where sg denotes stop-gradient. In the forward pass, $m^{\mathrm{STE}} = m^{\mathrm{hard}}$; in the backward pass, gradients flow through m^{soft}. The Support Gate uses the continuous normalized response $\bar{R}$ as its surrogate. For Top-K_c selection, the Control scores are transformed into a sigmoid soft mask whose cutoff is adjusted to match the selection budget. The aggregation weights of selected factors remain fully differentiable. Consequently, the prediction, SIGReg, and action-decoding losses jointly update the visual backbone, Support prototypes, and Control-Factor Selector. The STE is a biased gradient approximation for discrete selection; the local stop-gradient in the equation above does not prevent end-to-end updates to the target encoder.

Joint objective. The loss weights in Eq. 5 are set to $\lambda_{\mathrm{sig}} = 0.09$, $\lambda_{\mathrm{sup}} = 1$, and $\lambda_{\mathrm{act}} = 0.5$. The action decoder recovers the action block from the encoded adjacent observations $[z_t; z_{t+1}]$ and receives neither the action input nor the predictor output. This auxiliary supervision is used only during training; neither distractor masks nor the action decoder are required at test time.