Toward Compositional Latent Action Interfaces for Generalizable Agents

Heejeong Nam [1] **Chandradithya S Jonnalagadda** [1] **Harshit Aggarwal** [1] **Eric Xu** [1]

Abstract

Latent Action Models (LAMs) learn action proxies from observation transitions, but they face a fundamental ambiguity in multi-object or distractor-rich scenes: without supervision, the model cannot determine which changes are caused by the controlled agent. When multiple transition sources and environment-specific factors are compressed into a single monolithic latent action, the resulting representation can become sensitive to distractors and may generalize poorly under distribution shift. In this paper, we introduce Observed Transition Factorization (OTF), a framework that represents each transition as a sparse composition of reusable observed-transition primitives, yielding a compositional latent action interface. Building on this representation, we propose OTF-LAM, a latent action model that constructs action-like latents from factorized transition primitives within the standard inverse–forward dynamics framework. Empirically, OTF primitives transfer zero-shot across controlled carrier shifts and cross-morphology DCS settings, support downstream policy learning through OTF-LAM, and provide a promising path toward decoder-free latent action modeling with frozen visual representations.

1. Introduction

World models (Ha & Schmidhuber, 2018) for planning and control are traditionally evaluated in settings where the actor and the action space are known, such as a robotic manipulator (Hansen et al., 2024; Hafner et al., 2025). In this regime, the model predicts the next state from the current state and a proprioceptive signal, and can therefore, at least implicitly, separate the controlled source of variation from autonomous background dynamics or moving distractors. Recently, motivated by the fact that most online videos do not provide frame-level action labels or proprioceptive signals, Latent Action Models (LAMs) (Schmidt & Jiang, 2024; Ye et al., 2025) have been proposed, aiming to learn action-like proxies directly from observation transitions. A typical LAM encodes an observation pair (x_t, x_{t+1}) into a latent action and uses it to predict x_{t+1} from x_t. However, without additional inductive biases, the model has no direct basis for deciding which observed changes should be attributed to the controlled agent or selected as relevant to the desired action abstraction. We refer to this problem as the *agent non-uniqueness of unsupervised action learning*, illustrated in Figure 1. This ambiguity is often hidden in controlled settings where non-agent elements are static (Yu et al., 2020; Mandlekar et al., 2021). Generally, however, a transition may mix agent motion with passive objects, articulated parts, camera/background dynamics, and distractors (Nikulin et al., 2025; Wang et al., 2025; Adnan et al., 2026; Klepach et al., 2026). Since the model observes only the visual consequences of state changes, the transition is several steps removed from the true cause and represents a mixture of transition effects. This suggests that unsupervised action learning is not merely an architectural problem.

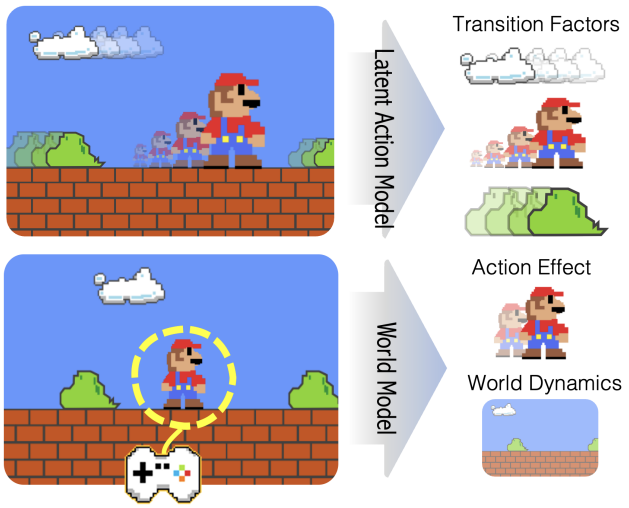


Figure 1. Motivation for agent non-uniqueness in observation-only latent action learning. Unlike action-conditioned world models, which receive the actor and action as part of the input, LAMs infer action-like latents only from visual transitions.

A natural response to this ambiguity is to treat it as an object

[1]Brown University. Correspondence to: Heejeong Nam <heejeong_nam@brown.edu>.

Accepted to the 2nd Workshop on Compositional Learning at ICML 2026, Seoul, South Korea. *Copyright 2026 by the author(s).*

selection problem (Wang et al., 2025; Adnan et al., 2026; Klepach et al., 2026), identifying the controllable entity to learn its latent action. However, relying on segmentation or object-centric priors limits scalability and imposes rigid assumptions. More importantly, objects are not always the appropriate unit of abstraction, as motion effects frequently manifest within, across, or independently of strict object boundaries. Recognizing the limitations of these predefined abstractions, we step back to reconsider the fundamental role of LAMs. Moving beyond the standard operational view of LAMs as simple transition bottlenecks (Schmidt & Jiang, 2024), we investigate what can be identified from observation-only transitions and what latent actions should ultimately represent.

We therefore revisit what can be identified from observation-only transitions and what latent actions should represent. We decompose the pathway from action to observation into three levels: the true underlying action, the induced physical state change, and the resulting visual transition effect in observational space. We suggests that, without additional assumptions, observed latent actions are better understood as proxies grounded in visible effects rather than direct estimates of true actions. Motivated by this view, we introduce **Observed Transition Factorization (OTF)**, a framework that encourages compositionality by decomposing observed transitions into reusable observed-transition primitives before compressing them into action-like latents. Rather than representing all frame changes with a single monolithic code, OTF models a transition as a sparse composition of localized visual effects. Notably, OTF does not recover exact motor commands or physical forces, but isolates the observed effects.

We instantiate OTF with a compositional VQ-style codebook of observed-transition primitives. Given an arbitrary motion representation of transition (e.g., gradient, velocity, optical flow), we patchify the transition and quantize local patch embeddings using a global codebook. Each code represents a reusable observed-transition primitive, while code-specific occupancy maps indicate where the primitive is active. This representation can be integrated into the standard LAM pipeline as a structured intermediate between inverse-dynamics inference and forward-dynamics prediction. Empirically, we study OTF for compositional latent action learning across both controlled and embodied settings. On Moving MNIST (Srivastava et al., 2015) and Distracting Control Suite (DCS) environments (Stone et al., 2021), OTF primitives transfer zero-shot across held-out visual carriers and cross-morphology shifts, indicating that the codebook captures reusable transition structure rather than digit- or embodiment-specific templates. We further evaluate OTF-LAM on downstream policy learning, where the learned transition primitives provide a useful latent action interface that is competitive with monolithic LAM baselines. Finally, we present OTF-LAM-Dino, a decoder-free variant that predicts future states in a frozen DINO representation space, suggesting a promising direction for combining reusable transition vocabularies with reusable visual prediction spaces.

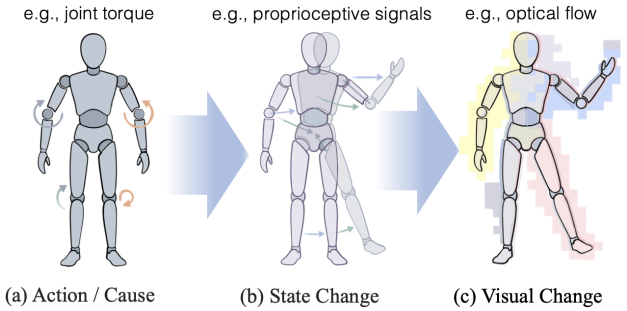


Figure 2. Conceptual decomposition from causes to observations. Our framework first learns a reusable vocabulary of visual change, and then uses environment-specific context to compose these effects into an action-like representation of the induced transition.

2. Unsupervised Latent Action Learning

LAMs trained from observation-only transitions do not recover actions directly, but representations of their visible consequences after being mediated by state, embodiment, dynamics, and rendering. As a result, the learned latent space is not necessarily action-equivalent.

Causes, State Changes, and Observed Effects

We view a transition as a chain of three stages:

$$\underbrace{a_t, \xi_t}_{\text{trainsition sources}} \xrightarrow{F_{\mathcal{E}}} \underbrace{\Delta s_t}_{\text{physical state change}} \xrightarrow{R_{\mathcal{E}}} \underbrace{\Delta x_t}_{\text{visual transition effect}}$$

as in Figure 2. Here, a_t denotes the controlled action, while ξ_t denotes uncontrolled transition sources such as camera motion, distractor motion, background dynamics, or passive environment changes. The environment $\mathcal{E}$ includes the embodiment, dynamics, camera, and other environment-specific factors. Let $F\mathcal{E}$ be the state transition function and $R_{\mathcal{E}}$ be the observation or rendering map. The physical state change may be written as

$$\Delta s_t = F_{\mathcal{E}}(s_t, a_t, \xi_t),$$

while the visual transition effect may be written as

$$\Delta x_t = R_{\mathcal{E}}(s_t, \Delta s_t, \xi_t).$$

Equivalently, the observed transition is generated by the composition

$$\Delta x_t = R_{\mathcal{E}}\big(s_t, F_{\mathcal{E}}(s_t, a_t, \xi_t), \xi_t\big). \tag{1}$$

Intuition. This setup clarifies why a latent action cannot generally recover the true action. The observed transition is generated by Eq. 1. For fixed observations, the map from a_t to Δx_t is generally not injective, and Δx_t may also contain changes not caused by a_t at all. Therefore, an observation-trained latent action can at best identify an equivalence class of causes and observed effects.

Reusable Structure in Observed Effects

Actions and physical state changes are often environment-specific. Actions depend on the control interface, such as torques or button inputs, while physical changes depend on state, embodiment, contact, and dynamics. Thus, neither level provides an obviously reusable primitive across heterogeneous embodiments. Observed visual effects offer a weaker but more transferable level of abstraction. They describe how changes appear in observational space such as local displacement, edge shifts, rotation-like residuals, contact deformation, or background drift. For instance,

`Mario moves right` `Kirby moves right`

may involve different body structures and dynamics, yet both can produce similar rightward displacement patterns in the observation space. This motivates using observed visual effects as the intermediate target for LAMs.

3. Observed-Transition-Factorized Latent Action Model (OTF-LAM)

A standard LAM (Schmidt & Jiang, 2024) with monolithic latent action learns $z_t = q(x_t, x_{t+1})$, directly compressing the transition into a single vector. We instead separate observed-transition from action abstraction

$$E_t = q_\phi(x_t, x_{t+1}), \qquad \text{and} \qquad z_t^{\text{act}} = h_\psi(E_t, x_t), \quad (2)$$

where, E_t denotes a structured set of observed-transition factors, while z_t^{act} is the compact action-like latent used for prediction. Our framework is trained in two stages. In Sec. 3.1, we pretrain the Observed-Transition Factorizer q_ϕ with a motion-space autoencoding objective. In Sec. 3.2, we freeze q_ϕ, and train a OTF-LAM on top of the extracted factors.

3.1. Observed Transition Factorization (OTF)

We construct the motion input o_t as a frame-level transition signal over a temporal stride τ, chosen to emphasize observable change while reducing dependence on RGB appearance, object identity, and environment-specific semantics. Its exact form is a design choice and described in Appendix D.1. Throughout, o_t denotes an observed effect rather than a physical state difference. The vocabulary stage

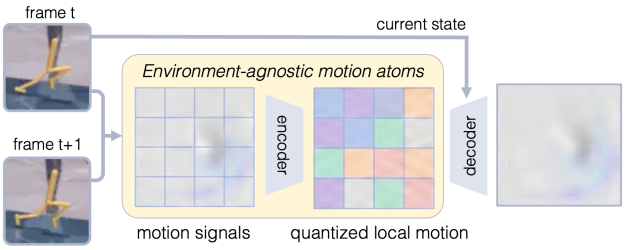


Figure 3. Observed-transition vocabulary learning with a VQ-VAE. Motion-centered transition inputs are patchified, quantized into a shared codebook of reusable observed-transition primitives, and decoded with reference-frame conditioning to reconstruct the observed-transition.

does not infer latent actions or assign changes to action-level causes. It instead learns a shared codebook of K reusable observed-transition primitives for representing o_t, capturing recurring visual change patterns such as local displacement, edge shifts, rotation-like residuals, contact deformation, background drift, or articulated part motion.

Patchwise vector quantization. Given $o_t \in \mathbb{R}^{C\times H\times W}$, we partition it into P non-overlapping spatial patches and encode each patch with a shallow MLP into a token $f_{t,i} \in \mathbb{R}^D$. Each token is then assigned to its nearest codebook entry using top-1 nearest-neighbor quantization

$$k_{t,i}^* = \arg\min_{k\in\{1,\dots,K\}} \|f_{t,i} - c^{(k)}\|_2^2, \quad q_{t,i} = c^{(k_{t,i}^*)}. \quad (3)$$

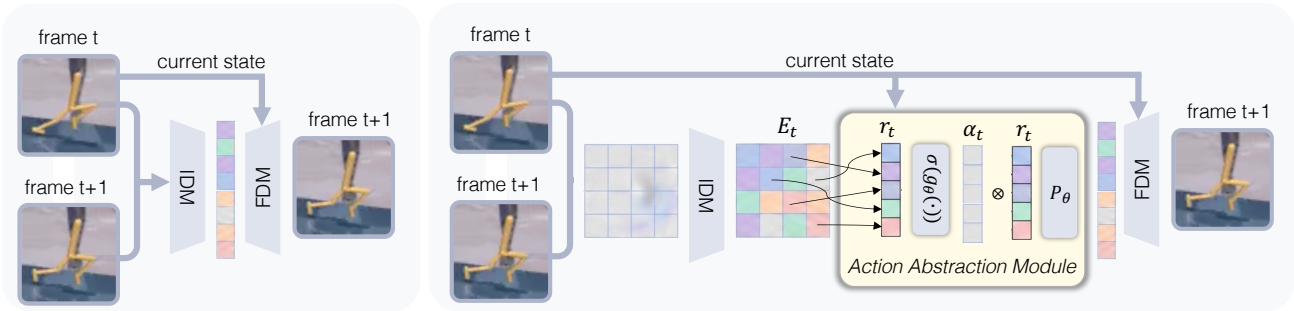
Note that this quantization should be interpreted as discretizing local observed effects rather than pure motion like optical flow. Even when o_t suppresses object-level semantics, patch tokens may retain carrier-specific cues such as texture, contrast, local geometry, and occlusion.

Intuition. Patch size controls the locality of the observed-transition primitives. Smaller patches reduce appearance and object-identity leakage, but may not contain enough context to infer a stable motion effect. Larger patches provide richer motion evidence, but can mix multiple transition sources and become tied to local carrier appearance. A more detailed discussion is provided in Appendix E.2.

From the patch-level assignments, we derive for each code k an occupancy map $M_t^{(k)}$, denoting the spatial support or occupancy of code k in the transition, and an activation strength $w_t^{(k)}$. Together these form the transition

$$E_t = \{e_{t,k}\}_{k=1}^K, \quad e_{t,k} = \big(c^{(k)}, M_t^{(k)}, w_t^{(k)}\big). \quad (4)$$

Thus, E_t records which observed-transition primitives are active in an observation transition, where they appear, and how strongly they are expressed, without yet determining which of them are action-relevant. In later stages, this



(a) LAPO

(b) LAM with Factorized Motion

Figure 4. **Latent-action learning on top of the pretrained observed-transition vocabulary.** Given the extracted observed-transition factors E_t and the current frame x_t, the model builds state-aware factor tokens, scores them with a relevance gate, aggregates the selected factors into a compact action-like latent z_t^{act}, and uses a reference-conditioned decoder to predict the next observation.

observed-transition vocabulary can be compressed or selected into action-like latents for downstream LAM training.

Training. Our training scheme follows Figure 3. We instantiate the observed-transition factorizer as a VQ-VAE trained to reconstruct the motion observation o_t rather than the future RGB frame. Each transition factor $e_{t,k} \in E_t$ contains the information associated with code k. We map each active factor to an aggregated factor embedding

$$h_{t,k} = \rho_\eta(e_{t,k})$$

, where ρ_η is a small shared network and k indexes a code observed in the current transition. Let $\mathcal{I}_t$ denote the set of active codes. The resulting set of factor embeddings is

$$H_t^{\text{set}} = \{h_{t,k}\}_{k \in \mathcal{I}_t}.$$

To preserve spatial support, we place these embeddings back onto the patch grid using the occupancy maps

$$H_t(u,v) = \sum_{k=1}^{K} M_t^{(k)}(u,v)\, h_{t,k}.$$

The decoder then combines the spatial factor map H_t with the current frame x_t, which provides appearance and support information, to reconstruct the motion observation

$$\hat{o}_t = D_\eta(H_t, x_t).$$

This encourages the codebook to represent local transition effects rather than object-specific appearance templates. OTF is trained with a motion reconstruction objective, standard VQ losses, and an orthogonality regularizer:

$$\mathcal{L}_{\text{vocab}} = \mathcal{L}_{\text{rec}} + \lambda_1 \mathcal{L}_{\text{code}} + \lambda_2 \mathcal{L}_{\text{commit}} + \lambda_3 \mathcal{L}_{\text{orth}},$$

where

$$\mathcal{L}_{\text{rec}} = \|\hat{o}_t - o_t\|_2^2, \tag{5}$$

$$\mathcal{L}_{\text{code}} = \sum_{i=1}^{P} \|\text{sg}(f_{t,i}) - q_{t,i}\|_2^2 \tag{6}$$

$$\mathcal{L}_{\text{commit}} = \sum_{i=1}^{P} \|f_{t,i} - \text{sg}(q_{t,i})\|_2^2, \tag{7}$$

$$\mathcal{L}_{\text{orth}} = \sum_{\substack{k,k' \in \mathcal{I}_t \\ k \neq k'}} \left(\hat{\mathbf{c}}^{(k)\top} \hat{\mathbf{c}}^{(k')}\right)^2. \tag{8}$$

Details of architecture and training schemes are provided in Appendix A.

3.2. OTF-LAM

After learning the OTF module, we freeze the factorizer and train a LAM on the extracted factor set E_t, as shown in Fig. 4. Our goal is to learn an action-like latent

$$z_t^{\text{act}} = h_\psi(E_t, x_t)$$

by selecting or compressing observed-transition factors for next-frame prediction.

Action abstraction. We first convert each factor in E_t into a state-aware factor token

$$r_{t,k} = \Gamma_\theta(c^{(k)}, M_t^{(k)}, w_t^{(k)}, x_t),$$

where Γ_θ embeds the code identity, occupancy, activation strength, and current-frame context. A gate network then assigns each factor a relevance weight $\alpha_{t,k} = \sigma(g_\theta(r_{t,k})) \in [0,1]$, with inactive codes masked out. The factor-level latent is obtained by a normalized gated average

$$z_t^{\text{fac}} = \frac{\sum_{k=1}^{K} \alpha_{t,k} r_{t,k}}{\sum_{k=1}^{K} \alpha_{t,k} + \epsilon}.$$

Finally, a linear projection produces the action-like latent $z_t^{\text{act}} = P_\theta(z_t^{\text{fac}})$. This module does not assume that the factorizer has identified the controlled agent. It only learns which observed-transition factors are useful for forming a compact latent for prediction.

Forward dynamics. We predict the next observation from the current frame and the latent action

$$\hat{x}_{t+\tau} = p_\theta(x_t, z_t^{\text{act}}).$$

Features from x_t provide appearance and layout information, while z_t^{act} modulates the decoder to provide transition information. The decoder may predict either the next frame directly or a residual update to the current frame, we use residual prediction by default.

Training. A full latent action model is trained with a next-frame prediction loss

$$\mathcal{L}_{\text{OTF-LAM}} = \|\hat{x}_{t+\tau} - x_{t+\tau}\|_2^2 .$$

The observed-transition vocabulary encoder and codebook remain frozen, and gradients are applied only to the action abstraction module and the forward dynamics model. Architectural details of the factor token construction, gating network, and decoder are provided in Appendix B.

JEPA-style OTF-LAM. We also explore a decoder-free JEPA-style variant, OTF-LAM-Dino. We map both the current and future observations into a latent representation space using a frozen DINOv2 encoder, and train the model to predict the future representation with z_t^{act}. This design keeps not only the motion vocabulary but also the visual encoder reusable across environments. Appendix F describes details of OTF-LAM-Dino architecture and training objective.

Intuition. OTF-LAM separates reusable transition primitives from the image decoder, but the decoder itself can still become environment-specific. OTF-LAM-Dino pushes this separation further by replacing pixel-space prediction with prediction in a frozen DINO representation space. In this variant, both the observed-transition vocabulary and the visual state encoder are fixed and reusable, leaving the learned model to focus on abstracting transition information into z_t^{act} and predicting its effect in representation space.

4. Experiments

All additional visualizations are in Appendix D, and dataset details are in Appendix C.

4.1. Reusability of Latent Action Primitives

We first use Distracting Control Suite (Stone et al., 2021) environment to evaluate the observed-transition primitives

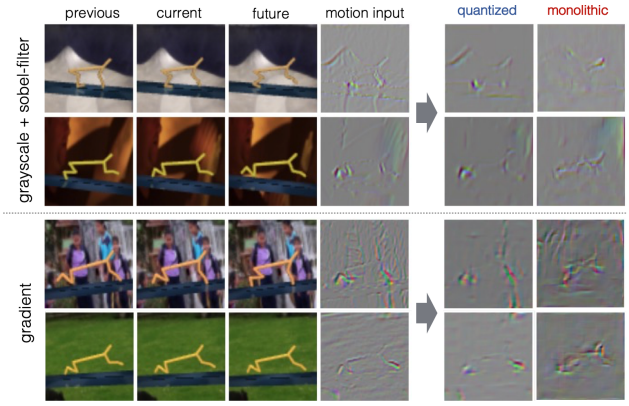


Figure 5. Cross-morphology transfer in DCS using only the pretrained observed-transition factorizer. The vocabulary is trained on `walker-run` and evaluated zero-shot on `cheetah-run`.

learned by the pretrained OTF tokenizer. A useful motion vocabulary should transfer across visual carriers and embodied morphologies, rather than overfitting to a single agent. To test this, the vocabulary is trained on `walker-run` and applied zero-shot to `cheetah-run` environment, with the codebook kept fixed. As a baseline, we use a monolithic VQ-VAE that encodes each transition into a single latent bottleneck with a comparable latent budget.

We also conduct a similar transfer study on Moving MNIST (Srivastava et al., 2015), which provides a more controlled setting for evaluating transfer across visual carriers. We use held-out digit classes to evaluate transfer, while keeping the underlying observed-transition primitives shared. We construct a training set containing digits from $\{0, 1, 2, 3, 4\}$ and a test set containing digits from $\{5, 6, 7, 8, 9\}$ where the vocabulary is evaluated on held-out digit identities at test time. Each sequence contains one or more digits moving on a blank canvas, with motion sampled independently of digit identity.

Results on transferability. We provide qualitative and quantitative results on transferability. Figure 5 shows that OTF reconstructs motion signals under a test-time embodiment shift across different DCS environments, and Figure 6 shows results on controlled carrier transfer on Moving MNIST dataset. Larger versions of these figures are provided in Appendix. From the reconstructed motion signals, we observe that OTF remains better aligned with the target transition. Note this difference can be visually subtle, since the reference and future frames are often close in pixel space. A model that copies the reference frame may therefore look plausible, even when it fails to incorporate the actual future transition. For this reason, the reconstructions should be compared against whether they reflect the target motion, not only whether they preserve a sharp reference appearance. The monolithic VQ-VAE baseline often shows

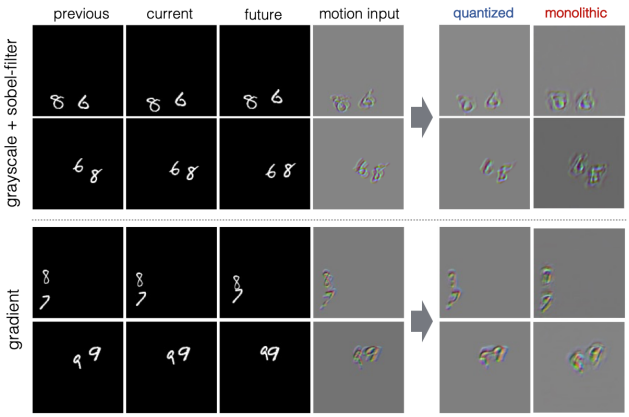


Figure 6. Controlled carrier transfer on Moving MNIST. The motion VQ-VAE is trained on digits `0,1,2,3,4` and evaluated zero-shot on held-out digits `5,6,7,8,9` with the same observed-transition primitives.

this failure mode, producing reconstructions closer to the reference frame than to the future frame. This suggests that its latent bottleneck entangles appearance or morphology-specific structure with motion information and falls back toward the static reference appearance under transfer. By contrast, OTF maintains a more usable reconstruction of the observed motion, indicating that its codebook captures observed-transition primitives that are not tied to a single digit class or embodiment.

Quantitatively, we measure the relative transfer degradation from the source to the target $\text{Drop} = \frac{\mathcal{E}_{\text{target}} - \mathcal{E}_{\text{source}}}{\mathcal{E}_{\text{source}}}$, where $\mathcal{E}_{\text{source}}$ and $\mathcal{E}_{\text{target}}$ denote reconstruction error on the training and transfer sets, respectively. A smaller drop indicates better reusability. As shown in Table 1, the OTF vocabulary shows substantially smaller degradation than the monolithic global embedding with the same input type and transform. These results indicate that the factorized codebook is less sensitive to the morphology shift, while the global monolithic bottleneck more strongly entangles motion reconstruction with source-domain visual structure.

4.2. OTF-LAM

To evaluate OTF-LAM, we train a policy for a downstream task: `cheetah-run` and `walker-run` from the DCS (Stone et al., 2021). First, OTF-LAM's latent motion space is distilled into a policy $\pi(z_t^{\text{act}} \mid x_t)$ via behavioral cloning. Next, an action decoder is trained on a small action-labeled dataset (32 trajectories) to translate these predicted latent observed-transition primitives into the true action space. Note that the current state is also passed to the action decoder to properly associate motions with their corresponding objects (see Fig. 7). We evaluate three existing LAM baselines and two OTF-LAM-specific variants. The OTF-LAM-specific variants are designed to isolate the contributions of motion processing and patchwise latent quantization.

Table 1. Quantitative evaluation of cross-morphology zero-shot transfer degradation in DCS environments under different motion inputs and frame-level transforms. Motion inputs are described in Appendix D.1.

	Motion Input		**MSE**		**Drop (%)**
	type	transform	train	transfered	
quantized	acc.	gradient	0.142	0.171	20.49%
		sobel	0.088	0.107	20.98%
	vel.	gradient	0.037	0.055	49.34%
		sobel	0.024	0.036	51.58%
monolithic	acc.	gradient	0.172	0.271	58.18%
		sobel	0.120	0.196	64.06%
	vel.	gradient	0.048	0.080	64.78%
		sobel	0.034	0.058	71.60%

- **LAPO** (Schmidt & Jiang, 2024): A standard monolithic latent action model with inverse dynamics model (IDM) and forward dynamics model (FDM).
- **FLAM** (Wang et al., 2025): A factorized latent action baseline that decomposes scene dynamics into separate factors, addressing the limitations of a single monolithic latent action. We train two different models with differnt number of slots (4 and 8).
- **HiLAM** (Kim et al., 2026): A hierarchical latent action baseline that uses a pretrained low-level LAM to extract latent action and aggregates them into higher-level latent skills.
- **Motion-LAPO**: a LAPO variant that receives motion inputs together with pixel frames, isolating the effect of explicit motion processing.
- **F-LAPO**: a LAPO variant with an OTF-LAM-like patch encoder and patchwise quantization, isolating the effect of patch-level latent quantization.

In Figure 8, we report LAPO-normalized downstream returns. For each seed, the policy is evaluated over 10 trajectories, and each model is run with three seeds. OTF-LAM does not outperform FLAM (Wang et al., 2025), which uses a stronger factorized scene-dynamics inductive bias. However, OTF-LAM remains competitive with monolithic LAM baselines while using motion-centered inputs and a compositional VQ-style codebook to suppress distractor-entangled transition factors before action abstraction. These results support our hypothesis that factorizing observed transitions can provide a useful latent action interface under distractor-induced ambiguity, while also showing that explicit scene-level factorization remains a strong advantage

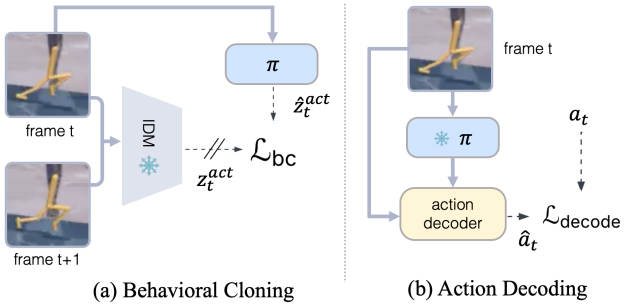


Figure 7. **Policy Training.** The latent motion space is distilled into a policy via behavioral cloning (left). An action decoder then maps these primitives to ground-truth actions a_t, utilizing the pixel frame to associate motions with specific objects (right).

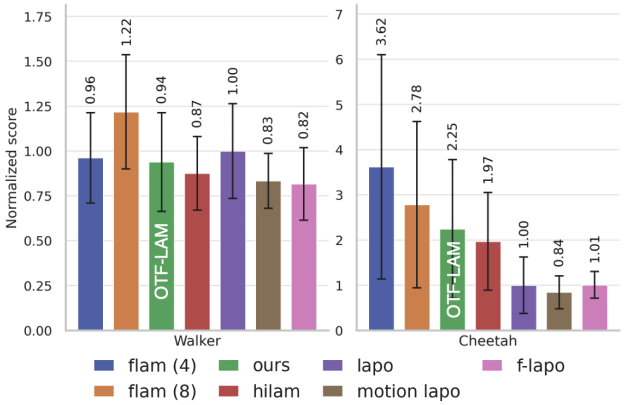


Figure 8. Bar chart comparing the relative mean return of latent action model variants on the DCS `cheetah-run` downstream policy evaluation task. Returns are normalized by the LAPO baseline. Error bar is computed as the RMS of standard deviations per seed.

when available. OTF-LAM performance is reported with its best-performing K.

Effect of Motion Vocabulary Size. We study how the size of the observed-transition vocabulary affects downstream policy performance. Figure 9 reports evaluation returns for OTF-LAM with different codebook sizes K on `cheetah-run` and `walker-run`, averaged by 3 seeds, where each seed evaluates 10 trajectories.

On `cheetah-run`, performance improves consistently as K increases,. This suggests that a larger vocabulary provides a more expressive set of transition primitives for this environment. On `walker-run`, however, performance is relatively stable across codebook sizes, with all variants achieving similar returns. This indicates that the required transition structure in `walker-run` is captured even by smaller vocabularies, or that downstream control is less sensitive to additional motion primitives in this setting. Overall, increasing K improves expressivity, but the benefit is

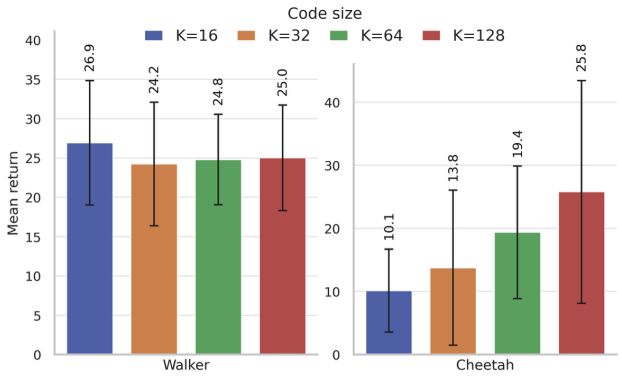


Figure 9. Downstream evaluation of OTF-LAM with different motion vocabulary sizes. Error bar is computed as the RMS of standard deviations per seed.

Table 2. Preliminary downstream evaluation of decoder-based OTF-LAM and OTF-LAM-Dino on `cheetah-run`. OTF-LAM-Dino uses a frozen DINO representation-space predictor instead of an image-space decoder. The OTF-LAM-Dino results are from a single seed.

Method	Code Size (K)	Mean Return	std.
OTF-LAM	16	10.142	6.574
	32	13.764	12.272
OTF-LAM-Dino	16	14.526	9.037
	32	14.022	4.904

environment-dependent. We therefore view codebook size as an environment-dependent capacity parameter rather than a monotonic source of improvement, and leave a more systematic study of this trade-off to future work.

4.3. Downstream Performance of OTF-LAM-Dino

We also evaluate OTF-LAM-Dino on downstream control. Unlike the decoder-based OTF-LAM, this variant predicts future states in a frozen DINO representation space and keeps the visual encoder fixed. This provides a more reusable prediction interface, since both the motion vocabulary and the visual state encoder are shared across environments. Table 2 reports preliminary results on `cheetah-run`. OTF-LAM-Dino obtains higher mean returns than the decoder-based OTF-LAM at both tested codebook sizes. Due to limited computational resources, we do not report the full sweep for OTF-LAM-Dino. The current OTF-LAM-Dino results are from a single seed and should be interpreted as preliminary. Nevertheless, we view this direction as a useful starting point for studying how general latent action models can become when both the transition vocabulary and the visual prediction space are made reusable, and leave a fuller investigation to future work.

5. Related Works

Factorized Motion and observed-transition Representations. Recent works feature video autoencoding methods to learn decoupled representations that can separate temporal motion from spatial information, which can be used for video reconstruction and generation (Shen et al., 2025; Yin et al., 2025; Liu et al., 2025). However, their primary objective is video reconstruction or generation and not the decomposition of multiple simultaneously moving sources into an abstraction for action-relevant factors. Other prior works have approached motion factorization through independent motion segmentation where the goal is to identify parts of a scene that move as either coherent or separately moving components (Meunier & Bouthemy, 2024; Wang et al., 2024; Choudhury et al., 2022). Ai et al. (2026) further show that motion structure can exist below the object level through dynamic-static disentanglement to separate independently moving parts from static geometry. Our focus is not on 3D reconstruction, but instead to learn reusable observed-transition primitives, which capture localized patterns of visual change caused by motions.

Latent Action Learning and Distractors. LAPO (Schmidt & Jiang, 2024) introduced this action-free learning framework for control from videos, while LAPA (Ye et al., 2025) scaled this idea by pretraining latent actions from large video datasets for VLA modeling. However, these models compress each transition into a single compact latent action. This can perform well in controlled settings where the agent is the main source of change, but becomes ambiguous in scenes with distractors where background/camera motion or other exogenous motions may be entangled with the latent action (Nikulin et al., 2025; Stone et al., 2021).

Nikulin et al. (2025) show that latent action learning can degrade under distractors, and that limited action supervision can improve performance. Other methods improve robustness by constraining or focusing the visual transition: LAOF (Bu et al., 2025) uses optical-flow constraints, Klepach et al. (2026) uses object-centric scene decomposition, and Adnan et al. (2026) uses pretrained segmentation masks to suppress background distractors. Recent work also studies latent action world models in more diverse in-the-wild videos (Garrido et al., 2026). In contrast, we factorize the observed transition into multiple motion-effect factors and learn to gate them with current-frame context. Several recent methods introduce structured latent action spaces. AC-LAM (Wei et al., 2026) uses additive component vectors, FLAM (Wang et al., 2025) decomposes dynamics into entity-level IDM/FDM modules, and LPWM (Daniel et al., 2026) models object-centric stochastic dynamics with particle-level latents. Our work instead factorizes transitions through patch-level activations of a shared observed-transition codebook.

Compositional and Quantized Latent Representations. VQ-VAE introduced learned discrete codebooks as reusable latent units, and LAPA uses a VQ-VAE-style objective to learn discrete latent actions from transitions (Van Den Oord et al., 2017; Ye et al., 2025). Compositional VQ-VAE-style representations have also been used to decompose complex human motion into reusable discrete components, for example in speech-driven holistic 3D motion generation (Yi et al., 2023) and music-conditioned dance video generation (Chen et al., 2025). Product quantization has also been used to learn compact discrete representations in visual and multimodal domains (Ma et al., 2024; El-Nouby et al., 2023). A complementary line of work studies compositionality through object-centric representations, where objects or slots serve as the basic units of decomposition (Locatello et al., 2020; Nam et al., 2026; Daniel & Tamar, 2022). In contrast, we build on the idea that codebooks can provide reusable components. This provides a compositional latent interface alternative to compact latent actions, where individual codes can be analyzed through their localized effects on the reconstructed transition.

6. Conclusion

We presented OTF-LAM, a compositional approach to latent action learning that factorizes observation transitions into reusable observed-transition primitives before composing them into action-like latents for prediction and control. Since observation-only LAMs see visual consequences rather than true actions, a single monolithic latent can entangle agent motion with distractors, camera dynamics, and object- or environment-specific appearance. OTF addresses this by learning a sparse vocabulary of localized transition primitives. Across Moving MNIST and DCS, these primitives transfer across held-out carriers and morphology shifts, suggesting that they capture reusable transition structure rather than object-specific templates. These results suggest a broader direction for scalable latent action learning: rather than compressing each video transition into a single action proxy, future LAMs can expose the compositional structure of observed effects and learn task- or embodiment-specific action abstractions on top. Studying this interface in larger-scale video pretraining, policy learning, and world-model objectives is a natural next step toward more generalizable latent action representations in distractor-rich environments.

Acknowledgement

We thank Chen Sun, Dan Haramati, and Randall Balestriero for helpful and meaningful discussions.

Impact Statement

This paper presents work whose goal is to improve the structure and generalization of observation-only latent action models. Progress in this direction may support downstream applications in robotics, video-based world modeling, and decision-making from unlabeled data, especially in settings with distractors or multiple sources of motion. The methods proposed here are primarily foundational and methodological, and we do not identify immediate negative societal consequences beyond those generally associated with the deployment of learned models.

References

Adnan, H., Jackson, M. T., and Zakharov, A. Segment to focus: Guiding latent action models in the presence of distractors, 2026. URL https://arxiv.org/abs/2602.02259.

Ai, H., Chang, W., Jiao, J., Leonardis, A., and Ofek, E. Articulation in motion: Prior-free part mobility analysis for articulated objects by dynamic-static disentanglement. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=mvKM40zDyn.

Bu, X., Lyu, J., Sun, F., Yang, R., Ma, Z., and Li, W. Laof: Robust latent action learning with optical flow constraints. *ArXiv*, abs/2511.16407, 2025. URL https://api.semanticscholar.org/CorpusID:283110282.

Chen, Z., Xu, H., Song, G., Xie, Y., Zhang, C., Chen, X., Wang, C., Chang, D., and Luo, L. X-dancer: Expressive music to human dance video generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10602–10611, 2025.

Choudhury, S., Karazija, L., Laina, I., Vedaldi, A., and Rupprecht, C. Guess What Moves: Unsupervised Video and Image Segmentation by Anticipating Motion. In *British Machine Vision Conference (BMVC)*, 2022.

Daniel, T. and Tamar, A. Unsupervised image representation learning with deep latent particles, 2022. URL https://arxiv.org/abs/2205.15821.

Daniel, T., Qi, C., Haramati, D., Zadeh, A., Li, C., Tamar, A., Pathak, D., and Held, D. Latent particle world models: Self-supervised object-centric stochastic dynamics modeling. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=lTaPtGiUUc.

El-Nouby, A., Muckley, M. J., Ullrich, K., Laptev, I., Verbeek, J., and Jegou, H. Image compression with product quantized masked image modeling. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=Z2L5d9ay4B.

Garrido, Q., Nagarajan, T., Terver, B., Ballas, N., LeCun, Y., and Rabbat, M. Learning latent action world models in the wild. *ArXiv*, abs/2601.05230, 2026. URL https://api.semanticscholar.org/CorpusID:284543722.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/2de5d16682c3c35007e4e92982f1a2ba-Paper.pdf.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. Mastering diverse control tasks through world models. *Nature*, 640 (8059):647–653, 2025.

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

Horn, B. K. and Schunck, B. G. Determining optical flow. *Artificial Intelligence*, 17 (1):185–203, 1981. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(81)90024-2. URL https://www.sciencedirect.com/science/article/pii/0004370281900242.

Kim, H., Pinto, L., and Kim, S. J. Hierarchical latent action model. In *ICLR 2026 the 2nd Workshop on World Models: Understanding, Modelling and Scaling*, 2026. URL https://openreview.net/forum?id=IUaiouMYXp.

Klepach, A., Nikulin, A., Zisman, I., Tarasov, D., Derevyagin, A., Polubarov, A., Lyubaykin, N., Kiselev, I., and Kurenkov, V. Object-centric latent action learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 22626–22634, 2026.

Liu, H., Sun, W., Zhang, Q., Di, D., Gong, B., Li, H., Wei, C., and Zou, C. Hi-vae: Efficient video autoencoding with global and detailed motion, 2025. URL https://arxiv.org/abs/2506.07136.

Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. Object-centric learning with slot attention. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

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

Lucas, B. D. and Kanade, T. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pp. 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

Ma, L., Ye, Y., Hong, F., Guzov, V., Jiang, Y., Postyeni, R., Pesqueira, L., Gamino, A., Baiyya, V., Kim, H. J., et al. Nymeria: A massive collection of multimodal egocentric daily motion in the wild. In *European Conference on Computer Vision*, pp. 445–465. Springer, 2024.

Mandlekar, A., Xu, D., Wong, J., Nasiriany, S., Wang, C., Kulkarni, R., Fei-Fei, L., Savarese, S., Zhu, Y., and Martín-Martín, R. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning (CoRL)*, 2021.

Meunier, E. and Bouthemy, P. Segmenting the motion components of a video: A long-term unsupervised model. *arXiv preprint arXiv:2310.01040*, 2024. URL https://arxiv.org/abs/2310.01040.

Nam, H., Lidec, Q. L., Maes, L., LeCun, Y., and Balestriero, R. Causal-jepa: Learning world models through object-level latent interventions, 2026. URL https://arxiv.org/abs/2602.11389.

Nikulin, A., Zisman, I., Tarasov, D., Lyubaykin, N., Polubarov, A., Kiselev, I., and Kurenkov, V. Latent action learning requires supervision in the presence of distractors. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://icml.cc/virtual/2025/poster/46574.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Schmidt, D. and Jiang, M. Learning to act without actions. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=rvUq3cxpDF.

Shen, C., Xu, L., Zhu, X., and Liu, G. Autoregressive video autoencoder with decoupled temporal and spatial context, 2025. URL https://arxiv.org/abs/2512.11293.

Sobel, I. and Feldman, G. M. An isotropic 3×3 image gradient operator. 1990. URL https://api.semanticscholar.org/CorpusID:59909525.

Srivastava, N., Mansimov, E., and Salakhutdinov, R. Unsupervised learning of video representations using lstms. In *Proceedings of the 32nd International Conference on Machine Learning*, pp. 843–852. PMLR, 2015. URL https://proceedings.mlr.press/v37/srivastava15.html.

Stone, A., Ramirez, O., Konolige, K., and Jonschkowski, R. The distracting control suite – a challenging benchmark for reinforcement learning from pixels, 2021. URL https://arxiv.org/abs/2101.02722.

Van Den Oord, A., Vinyals, O., et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

Wang, Z., Guo, J., and Daniilidis, K. Un-evimo: Unsupervised event-based independent motion segmentation. In *European Conference on Computer Vision*, pp. 228–245. Springer, 2024.

Wang, Z., Shi, C., Hu, J., Martín-Martín, R., and Stone, P. Flam: Scaling latent action world models with factorization. In *NeurIPS 2025 Workshop on Embodied World Models for Decision Making*, 2025.

Wei, H., Chen, X., Zhang, C., Pearce, T., Chen, J., Lamb, A., Zhao, L., and Bian, J. Learning additively compositional latent actions for embodied ai, 2026. URL https://arxiv.org/abs/2604.03340.

Ye, S., Jang, J., Jeon, B., Joo, S. J., Yang, J., Peng, B., Mandlekar, A., Tan, R., Chao, Y.-W., Lin, B. Y., Liden, L., Lee, K., Gao, J., Zettlemoyer, L., Fox, D., and Seo, M. Latent action pretraining from videos. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=VYOe2eBQeh.

Yi, H., Liang, H., Liu, Y., Cao, Q., Wen, Y., Bolkart, T., Tao, D., and Black, M. J. Generating holistic 3d human motion from speech. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 469–480, 2023.

Yin, X., Yuan, J., Hu, Z., Sun, W., Chen, J., Qiao, X., Li, H., and Sun, X. Deco-vae: Learning compact latents for video reconstruction via decoupled representation, 2025. URL https://arxiv.org/abs/2511.14530.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In Kaelbling, L. P., Kragic, D., and Sugiura, K. (eds.), *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 1094–1100. PMLR, 30 Oct–01 Nov 2020. URL https://proceedings.mlr.press/v100/yu20a.html.

A. Details of Learning the observed-transition Vocabulary

Encoder architecture. We train a motion-centric VQ-VAE to discretize local transition patterns into a shared vocabulary of observed-transition primitives. The input is a motion observation $o_t \in \mathbb{R}^{C\times H\times W}$ constructed from nearby frames, as described in the main text. The encoder partitions o_t into non-overlapping spatial patches and embeds each patch independently using a lightweight MLP patch encoder. We concatenate normalized spatial coordinates to each flattened patch representation before the MLP, so that the encoder has access to coarse patch location. The output is a grid of latent patch embeddings $\{f_{t,i}\}_{i=1}^{P}$, where each embedding corresponds to a local transition region.

Local code assignment. Each patch embedding is discretized using a learned codebook $\mathcal{C} = \{c^{(1)}, \dots, c^{(K)}\}$. Patch i is assigned to its nearest codebook entry under Euclidean distance, producing an index $k^*_{t,i}$ and a quantized embedding $q_{t,i} = c^{(k^*_{t,i})}$. We use the straight-through estimator to pass gradients from the decoder to the encoder through the quantized representation. To stabilize training, we use a short warmup phase in which quantization is disabled and the model behaves as a continuous autoencoder. After warmup, the codebook is initialized with k-means clustering over encoder patch embeddings collected from the training data, and discrete quantization is enabled.

Decoder input construction. The patch assignments are converted into the structured factor representation used in the main text. For each code k, we compute an occupancy map $M_t^{(k)}$ indicating which patches are assigned to the code, and a normalized usage weight $w_t^{(k)}$ equal to the fraction of assigned patches. Together with the codebook vector $c^{(k)}$, these define the observed-transition factor $e_{t,k} = (c^{(k)}, M_t^{(k)}, w_t^{(k)})$. Each factor is mapped to a descriptor $h_{t,k} = \rho_\eta(e_{t,k})$ using a shared MLP. The descriptors are then placed back onto the patch grid using the occupancy maps

$$H_t(u,v) = \sum_{k=1}^{K} M_t^{(k)}(u,v) h_{t,k}.$$

This produces a dense spatial motion feature map that preserves the factorized code identity and spatial support.

Decoder architecture. The decoder reconstructs the target motion observation from the spatial feature map H_t. It uses a convolutional decoding network with bilinear upsampling to produce an output at the original resolution. When reference conditioning is enabled, a separate convolutional encoder extracts spatial context features from the current frame x_t, and these features are combined with H_t before decoding. The reference pathway provides appearance and support information, while the codebook pathway provides the factorized transition information.

Training. The codebook model is trained to reconstruct the motion observation o_t rather than the future RGB frame. The objective combines a reconstruction loss with the standard VQ-VAE (Van Den Oord et al., 2017) codebook and commitment losses

$$\mathcal{L} = \mathcal{L}_{\text{rec}} + \lambda_{\text{code}}\mathcal{L}_{\text{code}} + \lambda_{\text{commit}}\mathcal{L}_{\text{commit}}. \tag{9}$$

When used, we additionally apply an orthogonality regularizer to active codebook entries to reduce redundancy in the learned vocabulary. The codebook is updated either by gradient-based optimization or by exponential moving average updates, depending on the experiment. For Exponential Moving Average (EMA)-updated codebooks, orthogonality primarily serves as a diagnostic for codebook diversity rather than a gradient-driven optimization objective. To reduce code collapse, rarely used codebook entries are periodically reinitialized using randomly sampled encoder embeddings. We train with AdamW (Loshchilov & Hutter, 2019) and gradient clipping. The resulting codebook defines the observed-transition vocabulary used by the latent action model.

B. Details of Learning the Latent Action Model

The notation follows the main paper. Let B denote the batch size, K the number of VQ codes, (h, w) the VQ-VAE patch grid, C the number of image channels, and (H, W) the image resolution. For convenience, we denote the stacked occupancy maps by

$$O_t \in \mathbb{R}^{B\times K\times h\times w},$$

where $O_{t,k} = M_t^{(k)}$ is the occupancy map of code k.

B.1. State Encoder

The state encoder extracts a spatial representation of the current observation conditioned on the observed-transition usage pattern. It takes the current frame $x_t \in \mathbb{R}^{B\times C\times H\times W}$ and the code usage vector $w_t \in \mathbb{R}^{B\times K}$ as input. The usage vector summarizes which observed-transition primitives are active in the observation transition, and is first projected into a compact conditioning vector

$$c_t^{\text{occ}} = W_{\text{occ}} w_t.$$

The image branch is a convolutional encoder with strided downsampling and residual blocks. Rather than concatenating the code usage vector to the image, we use c_t^{occ} to modulate intermediate visual features through FiLM conditioning (Perez et al., 2018) after each encoder stage. This allows the encoder to produce state features that are aware of the active observed-transition vocabulary while preserving the spatial structure of the current frame. The encoder outputs a spatial state feature map

$$s_t \in \mathbb{R}^{B\times d_s\times h\times w}.$$

When necessary, a 1×1 projection adjusts the channel dimension to d_s, and bilinear interpolation is used to match the target spatial resolution (h, w). We also compute a global state summary by spatial average pooling

$$\bar{s}_t = \frac{1}{hw}\sum_{i,j} s_{t,:,i,j}.$$

The spatial feature map s_t is used by the decoder to preserve layout information, while the pooled feature $\bar{s}_t$ provides a compact state representation for later conditioning modules.

B.2. Occupancy Encoder

The occupancy encoder converts the spatial support of each observed-transition code into a compact representation. Given the occupancy maps O_t, the encoder processes each code occupancy map independently and produces

$$u_t \in \mathbb{R}^{B\times K\times d_o}.$$

Each vector $u_{t,k}$ summarizes where code k is active in the current transition. By default, we use an MLP occupancy encoder. For each code k, the corresponding occupancy map is flattened and passed through a small network

$$u_{t,k} = f_{\text{occ}}(O_{t,k}; \bar{s}_t),$$

where $\bar{s}_t$ is the global state feature from the state encoder. We condition the hidden representation on $\bar{s}_t$ using FiLM, allowing the same occupancy pattern to be interpreted in the context of the current frame. This yields a compact state-aware embedding of each code's spatial support.

B.3. Factor Token Construction

The factor token construction module converts each observed-transition code into a state-aware token used by the action abstraction module. For code k, we first form a descriptor from the frozen codebook vector, its usage strength, and its encoded spatial occupancy:

$$d_{t,k} = [c^{(k)},\ w_{t,k},\ u_{t,k}].$$

Here, $c^{(k)}$ specifies the identity of the observed-transition primitive, $w_{t,k}$ indicates how strongly it is activated, and $u_{t,k}$ summarizes where it appears. We map this descriptor to a factor token

$$r_{t,k} = f_{\text{fac}}(d_{t,k}; \bar{s}_t, u_{t,k}),$$

where the embedding network is conditioned on the global state feature $\bar{s}_t$ and the occupancy embedding $u_{t,k}$ using FiLM. This conditioning allows the same codebook primitive to be interpreted differently depending on the current visual state and its spatial support. The resulting set $\{r_{t,k}\}_{k=1}^{K}$ is used by the action abstraction module.

B.4. Occupancy-Pooled Local State

The gate receives a local state vector for each factor. For code k, we compute this vector by occupancy-weighted average pooling over the spatial state feature map

$$\ell_{t,k} = \frac{\sum_{i,j} O_{t,k,i,j} s_{t,:,i,j}}{\max(\sum_{i,j} O_{t,k,i,j}, \epsilon)}.$$

If O_t and s_t do not share the same spatial size, the occupancy maps are resized to the state feature resolution using nearest-neighbor interpolation. In the standard pipeline, both are already defined on the VQ-VAE grid.

B.5. Gate Network

For each code k, the gate network takes the factor token, local state summary, and global state summary

$$g^{\text{in}}_{t,k} = [r_{t,k},\ \ell_{t,k},\ \bar{s}_t].$$

The hidden layers are FiLM-conditioned on

$$c^{\text{gate}}_{t,k} = [\bar{s}_t, u_{t,k}].$$

The gate network maps each factor representation to a scalar logit, which is converted into a continuous gating weight with a sigmoid activation

$$\alpha_{t,k} = \sigma(G_\theta(g^{\text{in}}_{t,k}; c^{\text{gate}}_{t,k})) \in [0, 1].$$

Codes with zero occupancy are masked out before aggregation.

B.6. Latent Action Aggregation

The model forms a factor-level latent by a normalized gated average over factor tokens

$$z^{\text{fac}}_t = \frac{\sum_{k=1}^{K} \alpha_{t,k} r_{t,k}}{\max(\sum_{k=1}^{K} \alpha_{t,k}, \epsilon)}.$$

The final action-like latent is obtained by a projection

$$z^{\text{act}}_t = P_\theta(z^{\text{fac}}_t).$$

The projection is the identity when the factor-token dimension already matches the desired latent dimension; otherwise, it is a linear layer.

B.7. Forward Decoder

The forward decoder predicts the next observation from the current-state features and the selected latent action. It receives the spatial state feature map s_t from the state encoder and the latent action z^{act}_t, and produces either a direct prediction of x_{t+1} or a residual update added to the current frame. To preserve spatial structure, we first project z^{act}_t into a low-resolution spatial grid

$$z^{\text{grid}}_t \in \mathbb{R}^{B \times d_g \times h \times w}.$$

This grid is concatenated with the state feature map s_t, allowing the decoder to combine the current visual layout with the transition information encoded in the latent action. The concatenated representation is then processed by a convolutional decoder with progressive upsampling. The latent action also modulates the decoder features through FiLM conditioning. After each upsampling stage, channel-wise scale and shift parameters are predicted from z^{act}_t and applied to the intermediate feature map

$$a \leftarrow (1 + \gamma(z^{\text{act}}_t)) \odot a + \beta(z^{\text{act}}_t).$$

This gives the latent action a direct mechanism for controlling the predicted transition while the spatial state features preserve appearance and layout. The decoder upsamples the feature map to the original image resolution and applies a convolutional output head. In residual mode, the output head predicts a residual that is added to x_t; in direct mode, it predicts the next frame directly. Unless otherwise specified, we use the residual formulation, which encourages the decoder to focus on transition-dependent changes rather than reconstructing the entire image from scratch.

C. Datasets

Distracting Control Suite (DCS) (Stone et al., 2021). We further evaluate our method on more complex datasets collected from environments based on the Distracting Control Suite, following the setup of Nikulin et al. (2025), where custom-difficulty datasets are constructed on top of DCS. We focus on the `cheetah-run` and `walker-run` environments, which contain articulated embodied motion together with visual distractors. We collect trajectories by using the pretrained expert policies from Nikulin et al. (2025). We collect 2000 trajectories each (2M transitions each) and only 1000 trajectories are used to the train the LAM while downstream evaluation is done using randomly sampled 32 trajectories from the other half.

These datasets allow us to test whether the learned observed-transition primitives remain useful beyond the controlled diagnostic setting, including zero-shot transfer across different embodiments.

Moving MNIST (Srivastava et al., 2015). We first evaluate our method on a controlled Moving MNIST benchmark, where digit identity and observed-transition primitive can be independently specified. This setting serves as a clean diagnostic environment for testing whether the learned codebook captures reusable motion/effect primitives rather than digit-specific visual templates. In particular, we use held-out digit classes to evaluate transfer across visual carriers while keeping the underlying observed-transition primitives shared. We construct a training set of 3,000 sequences containing digits from $\{0, 1, 2, 3, 4\}$ and a test set of 300 sequences containing digits from $\{5, 6, 7, 8, 9\}$.

D. Additional Visualizations

D.1. Motion-Centered Transition Inputs

To learn reusable observed-transition primitives effectively, the input should reduce semantic appearance information present in RGB frames while preserving the visual evidence of motion. We therefore consider several simple motion-centered input spaces. Specifically, we construct motion inputs by combining two temporal difference operators, which define the order of motion, with three frame-level transforms, which change the visual carrier through which motion is represented. Let ϕ denote a transformation applied independently to each frame. For first-order velocity inputs, we define

$$o_t^{\mathrm{vel},\phi} = \phi(x_{t+\tau}) - \phi(x_t). \tag{10}$$

For second-order acceleration inputs, we define

$$o_t^{\mathrm{acc},\phi} = \phi(x_{t+\tau}) - 2\phi(x_t) + \phi(x_{t-\tau}). \tag{11}$$

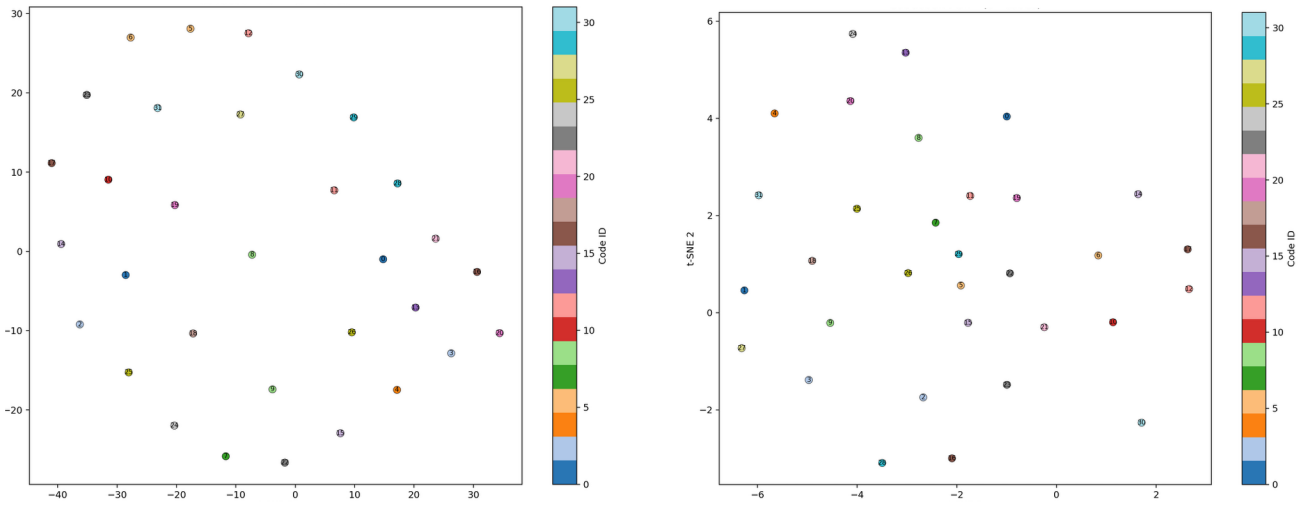
In all cases, the transform is applied before temporal differencing. We evaluate three choices of ϕ: the identity transform, Sobel (Sobel & Feldman, 1990) features after grayscale transformation, and image-gradient features. This yields six motion input variants: $\{\mathrm{velocity}, \mathrm{acceleration}\} \times \{\mathrm{none}, \mathrm{Sobel}, \mathrm{gradient}\}$. The identity transform captures raw frame differences, while Sobel and gradient transforms emphasize edge changes and local directional variation, respectively. Figure A1 shows some examples.

(a) Sobel (b) Gradient (c) None

Figure A1. Examples of motion-centered inputs produced by different frame-level transforms: Sobel, gradient, and none (identity).

D.2. Visualized Examples of OTF Module

We provide additional qualitative visualizations of the learned observed-transition vocabulary.



(a) MovingMNIST

(b) DCS Walker-run

Figure A2. Visualization of the learned codebook of observed-transition primitives on controlled Moving MNIST. Each point denotes one learned codebook vector, and the two-dimensional coordinates are obtained by applying t-SNE (van der Maaten & Hinton, 2008) to the code embeddings.

Distracting Control Suite (Stone et al., 2021). We train the observed-transition factorizer using the Sobel-transformed velocity input. Similarly to the Moving MNIST visualization, Fig. A2(b) shows the learned codebook after training the observed-transition factorizer on DCS samples. Fig. A3 shows example assignments on individual `walker-run` transitions, including the current frame, the motion-centered input, the local patch embeddings, and the assigned codebook entries.

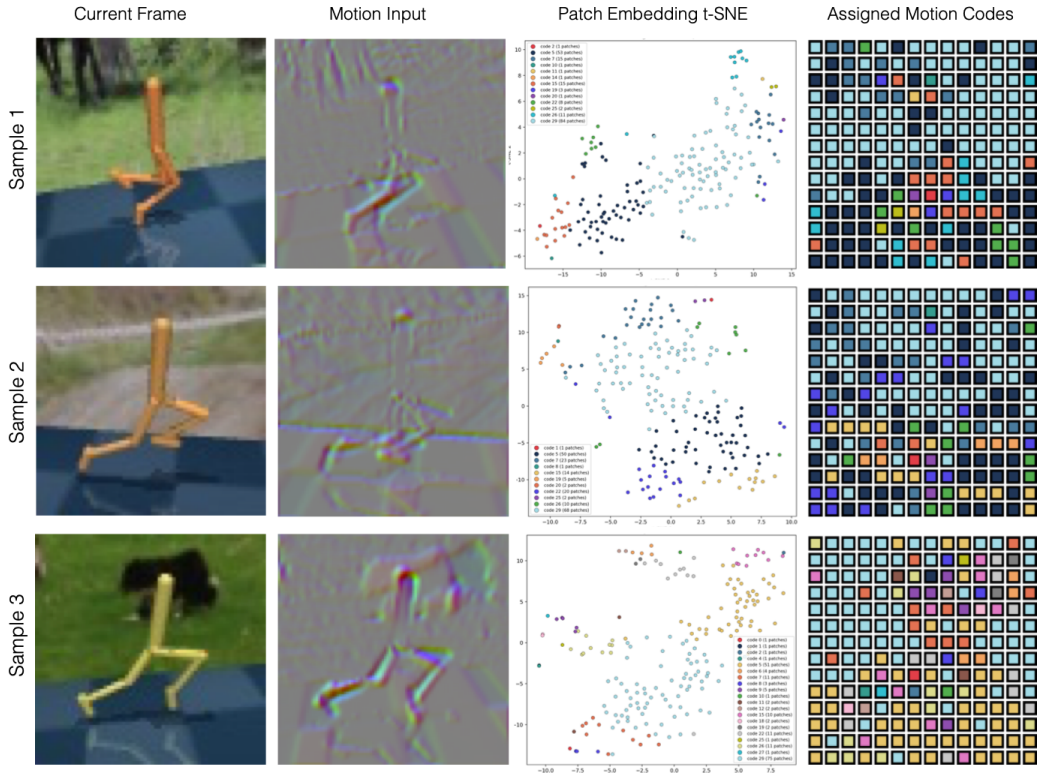


Figure A3. Example code assignments for walker-run transitions from Distracting Control Suite.

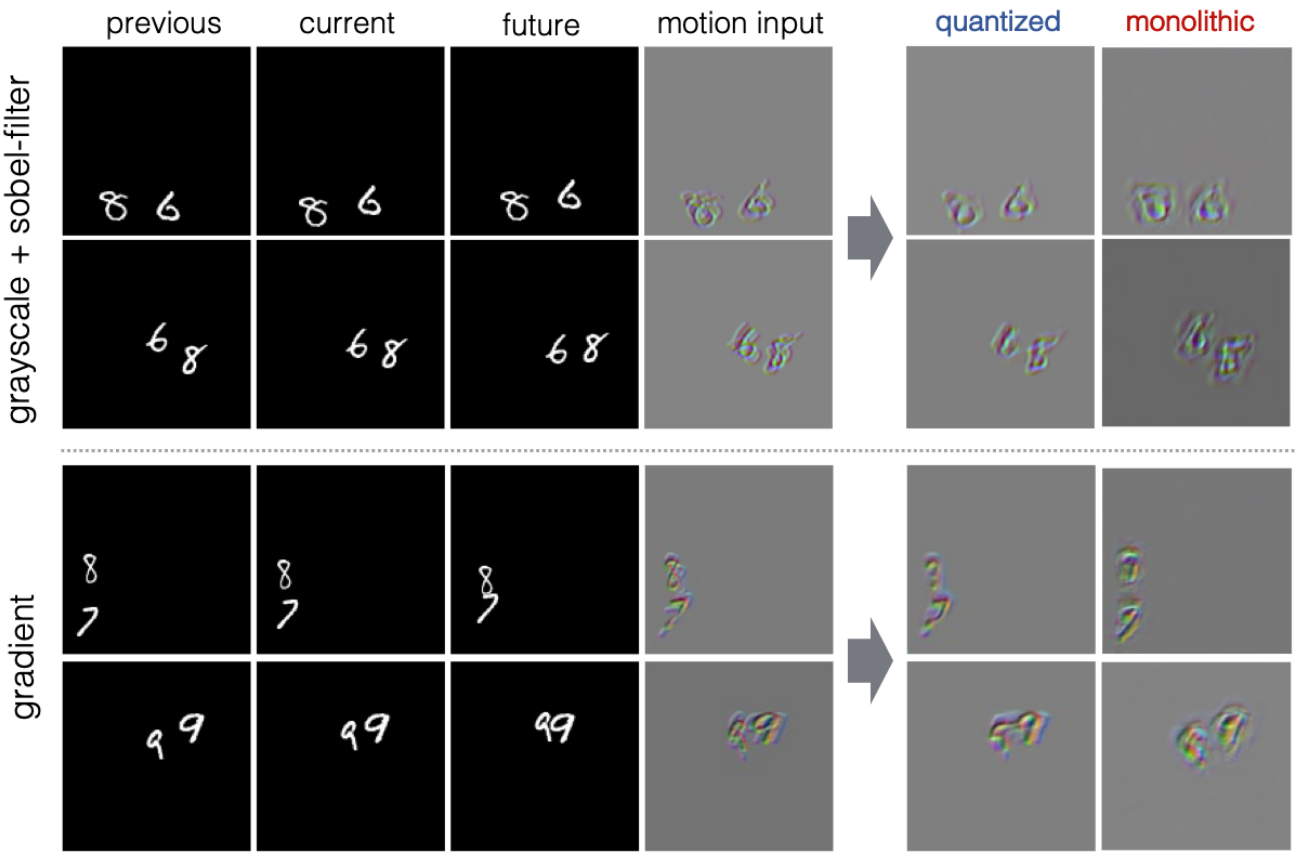


Figure A4. Zero-shot transfer experiment results on MovingMNIST.

MovingMNIST (Srivastava et al., 2015). We train the observed-transition factorizer using the Sobel-transformed acceleration input. To make the motion signal more pronounced, we use a temporal stride of two frames when constructing the transition input.

Fig. A2 (a) visualizes the learned codebook after training the factorizer described in Sec. 3.1. This visualization provides a qualitative view of how the discrete observed-transition primitives are organized in the learned vocabulary.

Fig. A6 shows example assignments on individual Moving MNIST samples. For each sample, we visualize the current frame, the motion-centered input, the t-SNE (van der Maaten & Hinton, 2008) projection of patch embeddings, and the corresponding assigned codebook entries on the patch grid or assigned observed-transition primitives on the patch grid. These examples illustrate how local transition patterns are mapped to discrete codebook primitives across different spatial regions.

Then we use Moving MNIST (Srivastava et al., 2015) as a controlled diagnostic for motion reusability. The VQ-VAE factorizer is trained on digits `{0,1,2,3,4}` and evaluated on held-out digits `{5,6,7,8,9}`, while the set of observed-transition primitives is shared across the two splits. Figure A4 shows result.

E. Observed Motion Signals and Patch Scale

E.1. Observed Motion Signals Are Not Pure Motion

The motion input used in this work is computed from image observations, for example by first- or second-order temporal differencing. Although this suppresses static appearance, it does not expose physical motion directly. A pixel-level velocity or acceleration signal is an *observed effect* of motion through the local visual content of the scene.

More formally, let $m_{t,p} \in \mathcal{M}$ denote the underlying local motion effect at patch p, such as translation, rotation, deformation,

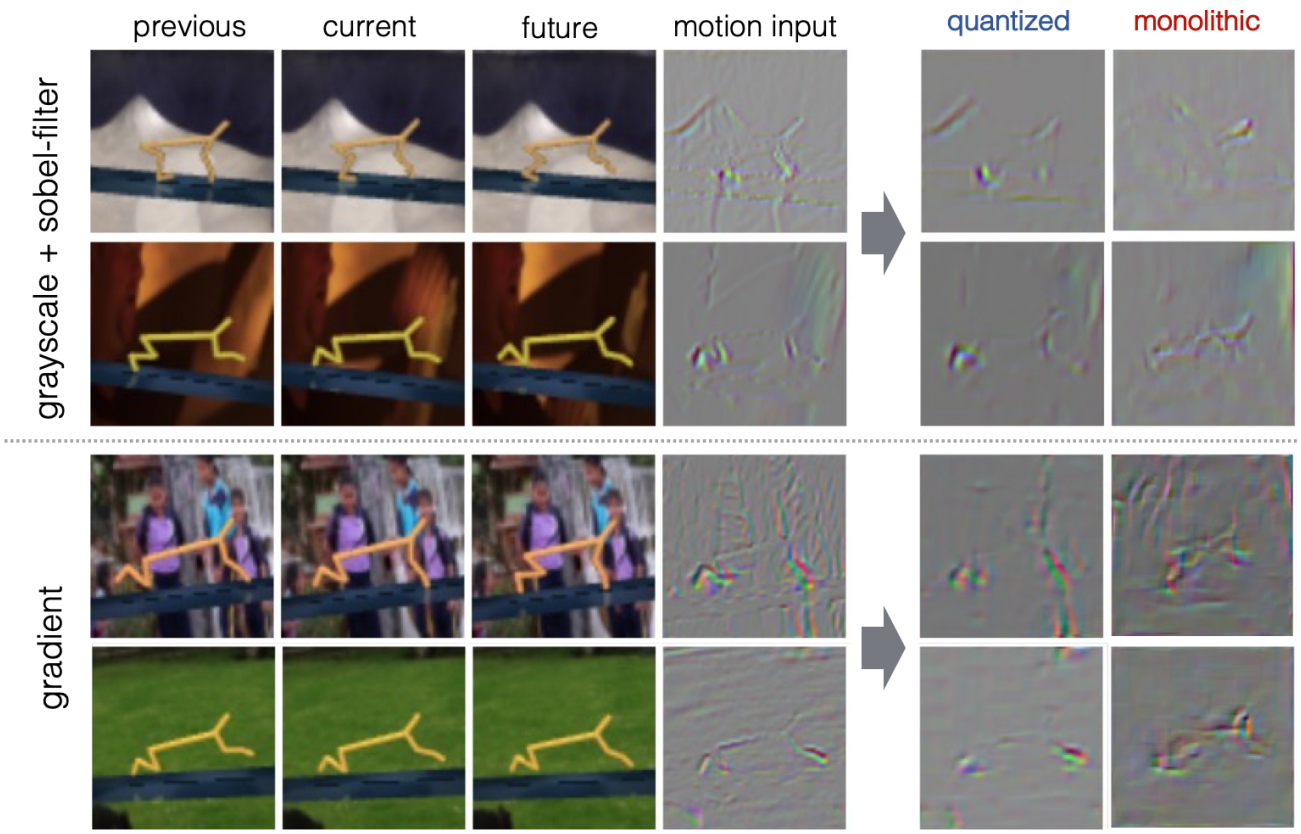


Figure A5. Enlarged version of Fig. **??** for readability.

or camera-induced displacement. Let $\xi_{t,p} \in \Xi$ denote the local visual carrier through which this motion is observed. This carrier may include local texture, contrast, edge orientation, color, depth discontinuities, occlusion boundaries, or other patch-level appearance statistics. The observed motion signal $y_{t,p}$ is then better understood as

$$y_{t,p} = \Psi(m_{t,p}, \xi_{t,p}), \tag{12}$$

for some image-formation-dependent map Ψ, rather than as a direct observation of $m_{t,p}$ alone. Thus, the codebook should not be interpreted as spanning the space of physical observed-transition primitives $\mathcal{M}$ in isolation. It is better viewed as learning from the space of observed motion effects

$$\mathcal{Y} = \{\Psi(m, \xi) : m \in \mathcal{M}, \xi \in \Xi\}. \tag{13}$$

This distinction is important for interpreting the learned codes. For example, the same rightward displacement can produce different observed signals depending on whether it occurs on a vertical edge, a textured background, a uniform region, or an articulated limb boundary. Conversely, patches with different appearance may still activate similar codes when their local transition patterns are sufficiently similar. Therefore, our codes should not be interpreted as pure geometric motion variables. They are more accurately viewed as reusable *observed-transition primitives*: local transition patterns that combine motion with the visual carrier through which that motion becomes visible.

This interpretation also clarifies the role of the reference-frame conditioning branch. The codebook is encouraged to represent transition effects, while the reference frame provides complementary static support information such as object shape and local appearance. In this sense, the method does not attempt to remove the visual carrier entirely. Rather, it limits the carrier information available to each local patch and avoids forcing the codebook to encode full object identity.

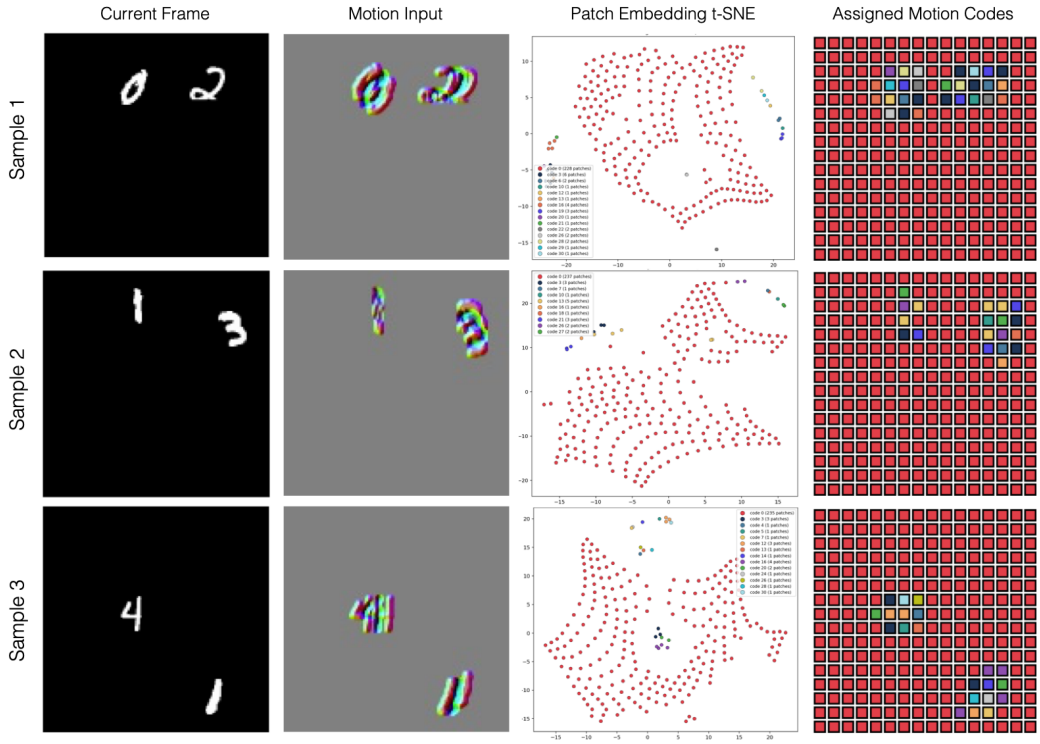


Figure A6. Example code assignments for controlled Moving MNIST transitions.

E.2. Patch Size and the Motion–Carrier Trade-off

Patch size controls the granularity at which observed transition effects are quantized. It therefore determines an important trade-off between motion locality and carrier ambiguity.

If patches are too small, each token contains only a limited visual neighborhood. This reduces leakage from object identity or large-scale appearance, but it can make local motion difficult to infer. Small patches are more sensitive to noise, texture aliasing, and the aperture problem (Horn & Schunck, 1981; Lucas & Kanade, 1981): a short edge fragment may not contain enough information to distinguish translation, rotation, or deformation. In such cases, the codebook may learn overly local residual patterns rather than stable motion-effect primitives.

If patches are too large, each token contains richer spatial context and may provide more reliable evidence for local motion. However, larger patches are also more likely to contain multiple motion sources, object-part identity, texture statistics, or foreground-background boundaries. As a result, the learned codes may become more appearance- or part-specific, approaching object-conditioned transition templates rather than reusable observed-transition primitives. In this regime, the effective code space may expand from a compact motion-effect basis toward a larger product space of motion and local carrier factors.

The desired patch scale is therefore task-dependent. A useful patch should be large enough to contain sufficient local structure for estimating a dominant transition effect, but small enough to avoid mixing multiple independent motion sources or encoding full object identity. In our setting, patchification is intended to preserve a direct local correspondence between the observed motion field and the latent code assignment. The patch size should therefore be interpreted as a design choice that controls the locality of the learned observed-transition primitives.

F. JEPA-style OTF-LAM

Here we describe a decoder-free variant, OTF-LAM-Dino, which replaces pixel-space prediction with prediction in a frozen visual representation space. The purpose of this variant is to test whether the latent action learned from observed-transition primitives can also condition a JEPA-style latent dynamics model, and furthur maintain downstream performance with

frozen DINOv2 encoder.

Latent visual state. Given an observation x_t, we encode it with a frozen DINOv2 encoder:

$$S_t = \mathrm{DINO}(x_t), \qquad S_{t+\tau} = \mathrm{sg}(\mathrm{DINO}(x_{t+\tau})),$$

where S_t denotes the patch-token representation and $\mathrm{sg}(\cdot)$ stops gradients through the future target. The DINO encoder is kept fixed throughout training. Thus, unlike the pixel decoder in the main OTF-LAM model, both the state representation and the prediction target are defined by a reusable pretrained visual encoder.

Latent action from observed-transition primitives. OTF-LAM-Dino uses the same pretrained OTF tokenizer as the main model. Given the transition pair $(x_t, x_{t+\tau})$, the frozen OTF encoder and codebook produce a structured set of observed-transition primitives,

$$E_t = (c_k, M_t^{(k)}, w_t^{(k)})k = 1^K.$$

An action abstraction module then aggregates these local transition primitives, conditioned on the current DINO state, into a compact latent action:

$$z_t^{\mathrm{act}} = h\psi(E_t, S_t).$$

The OTF tokenizer remains frozen by default, so gradients are applied only to the action abstraction module and the latent dynamics predictor.

JEPA-style forward dynamics. Instead of reconstructing the future frame, OTF-LAM-Dino predicts the future DINO representation:

$$\hat{S}_{t+\tau} = P_\theta(S_t, z_t^{\mathrm{act}}).$$

The predictor receives the current DINO patch tokens and a global latent action token derived from z_t^{act}. In the main variant, local observed-transition tokens are not directly passed to the predictor; this keeps z_t^{act} as the transition bottleneck. The latent action token is injected into the predictor at multiple layers, so the model cannot ignore it after the input layer.

Training objective. The model is trained with a latent prediction loss in frozen DINO feature space:

$$\mathcal{L}_{\text{OTF-LAM-Dino}} = \left|\hat{S}_{t+\tau} - S_{t+\tau}\right|_2^2.$$

Discussion. OTF-LAM-Dino follows the same latent action principle as OTF-LAM, but evaluates it in a different prediction space. The decoder-based model asks whether z_t^{act} can support future observation reconstruction, while OTF-LAM-Dino asks whether the same type of latent action can support future prediction in a reusable representation space. This variant is therefore complementary to the main model: it removes the pixel decoder, but still relies on the OTF vocabulary to construct the latent action from local observed-transition primitives.