# PATH COMPATIBILITY: PERSISTENT ANCHORS FOR CONTINUAL VLA FINE-TUNING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Pretrained vision–language–action (VLA) policies provide a foundation for robots to acquire new skills throughout deployment. Continual fine-tuning, however, changes the perception-to-action path and can disrupt previously learned behavior. Distillation from a previous-task teacher uses a reference that may already have drifted from the response learned when an old task was acquired. We introduce *Path Compatibility* (**PathComp**), which preserves old responses together with the interface inputs that produced them. After learning each task, PathComp caches modal tokens together with their fused latents and action distributions. During later tasks, the current fusion module and action head reproduce these acquisition-time responses from the cached tokens, while a separate modal anchor constrains the encoders on stored observations. This construction provides a fixed downstream reference while accounting for upstream drift, without a rolling teacher. We evaluate PathComp with a CLIP–LoRA–GMM policy on LIBERO-Object, LIBERO-Goal and LIBERO-Spatial, spanning changes in manipulated objects, task goals and spatial relations. Across these suites, it reduces final held-out action MSE by 21.5–22.6% relative to the strongest evaluated prior-work baseline under the shared policy and offline protocol in each suite. On LIBERO-Object, a shared-design target-time control and component ablations support the use of acquisition-time interface relations for retaining learned responses as the policy adapts. Code and checkpoints will be made public after acceptance.

## 1 INTRODUCTION

Pretrained vision–language–action (VLA) policies provide a foundation for robots that must keep acquiring new manipulation skills (Brohan et al., 2023; Kim et al., 2025; Octo Model Team et al., 2024; Black et al., 2025). A deployed robot also needs to retain the skills learned earlier: adapting to a new object or instruction should not undo familiar behavior. Continual fine-tuning brings these requirements together, asking a shared policy to learn from a stream of demonstrations while preserving its responses on past tasks (De Lange et al., 2022; Zhou et al., 2024; Wang et al., 2025).

The difficulty extends across the perception-to-action path. As the encoders, temporal fusion module and action head adapt, an old observation can produce different modal tokens, fused latents and actions. We call these changes *path drift*. A previous-task teacher constrains the current policy against its most recent predecessor (Li & Hoiem, 2018; Roy et al., 2025; 2026), but that predecessor already reflects intervening updates. Matching it need not recover the response learned when an old task was acquired. A stable output target addresses this changing reference; the remaining question is which inputs the adapting downstream modules should receive when matching that target. We use *path compatibility* to mean preserving these acquisition-time input–response relations as the surrounding policy adapts.

Our method, Path Compatibility (PathComp), preserves acquisition-time input–response relations at the policy's interfaces. After learning a task, it records modal tokens, their fused latents and action distributions on selected demonstration windows. During later tasks, the current fusion module and head act on those cached tokens and match the recorded responses. The same tokens also serve as targets for a modal constraint on stored observations. They therefore provide both a fixed input for downstream retention and a reference for changes in the adapting encoders. This separates the two sources of drift while keeping the target tied to task acquisition, without maintaining a rolling teacher.



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

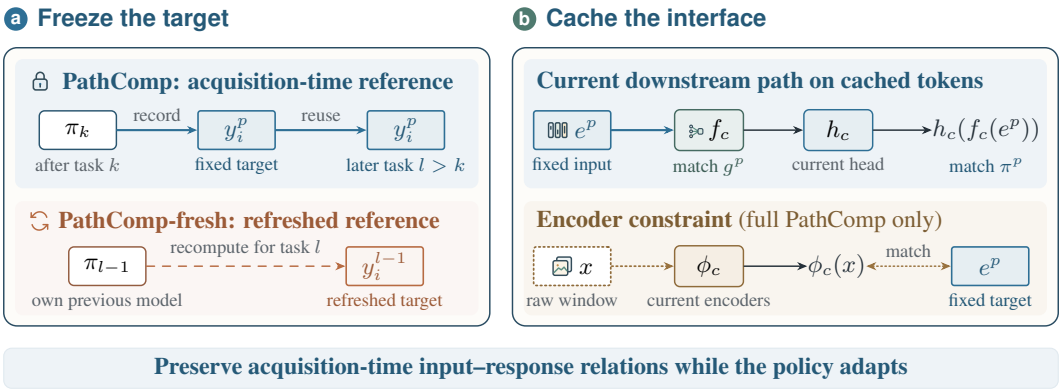


Figure 1: Two choices define an input–response anchor. **(a)** PathComp records targets after task $k$ and reuses them unchanged; PathComp-fresh uses its own previous-task checkpoint. **(b)** Cached tokens enter the current fusion module and head; stored latents and action distributions are targets. Full PathComp also constrains the encoders on raw windows; PathComp-RF (raw-free) omits that branch.

Persistent output targets and cached features each have established precedents. DER and CLEAR preserve acquisition-time outputs on raw replay inputs (Buzzega et al., 2020; Rolnick et al., 2019), while latent replay and multimodal latent replay reuse stored features with supervised labels (Pellegrini et al., 2020; Hayes et al., 2020; Yu et al., 2026). Teacher-based multimodal methods instead refresh their reference from the previous task (Roy et al., 2025; He et al., 2026). PathComp uses persistent inputs and responses to constrain the downstream composition, together with a separate constraint on the adapting encoders. Full PathComp retains raw windows for that modal constraint; its raw-free variant removes the constraint and exposes the storage–retention trade-off.

We evaluate this design on LIBERO-Object, LIBERO-Goal and LIBERO-Spatial (Liu et al., 2023), covering variation in manipulated objects, task goals and spatial relations. Using a shared CLIP–LoRA–GMM policy and protocol, PathComp reduces final held-out action MSE by 21.5–22.6% against the strongest evaluated prior-work baseline in each suite. Object provides the primary three-seed comparison; Goal and Spatial test the same PathComp constants against an M2Distill adaptation with two seeds each. A previous-task-target control built with the same window schedule, loss definitions, weighting rule and cached-input pathway yields higher Object error. Component ablations show that combining policy and feature constraints improves retention, while the raw-free comparison measures the modal constraint's benefit and storage cost. These results support the use of persistent interface anchors in the evaluated offline continual action-prediction setting.

We make two contributions:

1. **Persistent interface anchors.** We introduce a retention construction that preserves acquisition-time responses on cached downstream inputs while separately constraining encoder drift. A path decomposition explains the roles of these constraints and the raw-free variant.
2. **Evidence across task variations and design choices.** We report lower action error across three LIBERO suites, test the target-time rule with a shared-design control and examine component contributions, likelihood and storage to characterize when the method is effective.

## 2 Related Work

**When targets are computed.** Retention methods differ in when they construct the reference that later models must match. LwF, iCaRL and PODNet distill from a frozen copy of the previous model (Li & Hoiem, 2018; Rebuffi et al., 2017; Douillard et al., 2020); for lifelong imitation and VLAs, so do SPREAD, Info-VLA and M2Distill, which already distills every modality and the GMM policy (Roy et al., 2026; Zhao et al., 2026; Roy et al., 2025). DER, CLEAR and function-space methods instead match outputs recorded when a sample was stored or its task ended (Buzzega et al., 2020; Rolnick et al., 2019; Titsias et al., 2020; Pan et al., 2020; Benjamin et al., 2019), and X-DER revises such targets as new classes arrive (Boschini et al., 2023).

**Replay from cached features.** Latent replay and REMIND replay stored intermediate features with labels and keep the layers below frozen or slow, so that the features stay valid (Pellegrini et al., 2020; Hayes et al., 2020). MLR stores frozen-encoder multimodal latents with labels on LIBERO (Yu et al., 2026), and FAGR replays features generated from stored cluster statistics (Yang et al., 2025). LifelongVLA distills from the previous-task snapshot on cached prefix tokens (He et al., 2026), the rolling-target regime that our fresh-target control adopts. Feature adaptation and drift compensation instead correct stored features or prototypes for a backbone that keeps training (Iscen et al., 2020; Yu et al., 2020).

**Continual VLA adaptation.** Isolation and growth methods add adapters, skills or experts (Liu et al., 2024; Wan et al., 2024; Lee et al., 2024; Lei et al., 2026; Römer et al., 2026; Zhang et al., 2026) or route among experts without extra parameters (Wu et al., 2025); PathComp trains one shared set of parameters. O-LoRA and InfLoRA store no replay data and confine each task's LoRA update to a subspace that interferes little with earlier tasks (Wang et al., 2023; Liang & Li, 2024), while one recent study reports limited forgetting under sequential LoRA fine-tuning with on-policy RL (Hu et al., 2026). Other studies find strong retention from replay in pretrained VLAs, with outcomes depending on pretraining, replay frequency and action normalization (Liu et al., 2026; Zhu et al., 2026). Our controlled comparison uses a CLIP–LoRA–GMM policy with pretrained vision–language features. PHASER balances replay across task phases (Chen et al., 2026), and Memory Anchors are raw samples chosen by representational conflict (Du et al., 2026). PathComp's confidence weights and per-task quota build on selective constraints, such as FROMP's memorable points near the decision boundary (Pan et al., 2020) and SPREAD's top-$M$ confident actions (Roy et al., 2026), and on balanced memory (Chaudhry et al., 2019; Chen et al., 2026).

## 3 METHOD

PathComp preserves an acquisition-time input–response relation along the adapting policy. Cached modal tokens fix the inputs on which the current downstream modules match stored responses; a separate modal anchor constrains the encoders on stored observations (Figure 2). The cached tokens serve both as encoder targets and as downstream inputs, so each constraint addresses a distinct part of the adapting computation.

### 3.1 PROBLEM SETUP

Task $k$ of a stream of $K$ imitation-learning tasks provides demonstrations $\mathcal{D}^{(k)}$ of trajectories $\{(o_t, a_t)\}_t$ with an instruction $\iota$, where $o_t = (v_t, q_t)$ holds multi-view RGB images $v_t$ and the proprioceptive state $q_t$. On the context window $s_t = o_{t-L_o+1:t}$, the policy computes modal tokens $e_t$, a fused latent $g_t$ and an action distribution:

$$e_t = [\phi_v(v_t), \phi_\iota(\iota), \phi_s(q_t)], \qquad g_t = f_\psi(e_{t-L_o+1:t}), \qquad \pi_\theta(\cdot \mid s_t, \iota) = h_\omega(g_t). \tag{1}$$

The encoders $\phi$, fusion module $f$ and distribution head $h$ define the interfaces used by PathComp; $\theta$ collects their parameters. In our instantiation, $\phi_v$ is a frozen CLIP ViT-B/16 (Radford et al., 2021) with trainable LoRA adapters (Hu et al., 2022), $\phi_\iota$ projects a frozen CLIP text embedding, and $h_\omega$ outputs a Gaussian mixture (GMM). Other modules are initialized from scratch and trained; the policy has no action-pretrained checkpoint. Below, subscript $c$ denotes the current maps, subscript $k$ their state after task $k$, and superscript $p$ a cached acquisition-time value. Retention methods add a regularizer to the behavior-cloning loss $\mathcal{L}_{\mathrm{BC}}^{(k)} = \mathbb{E}_{\mathcal{D}^{(k)}}[-\log \pi_\theta(a_t \mid s_t, \iota)]$; those we compare share the form

$$\mathcal{L} = \mathcal{L}_{\mathrm{BC}}^{(k)} + \frac{1}{|\mathcal{P}|} \sum_{i \in \mathcal{P}} w_i \, d\big(S_c(u_i), y_i\big), \qquad \mathcal{P} \subset \mathcal{B}. \tag{2}$$

Here a minibatch $\mathcal{P}$ of a memory $\mathcal{B}$ (or of the current batch) supplies inputs $u_i$, targets $y_i$ and weights $w_i$. $S_c$ is the part of the current model that maps the replay input $u_i$ to the interface where $y_i$ lives, and $d$ is a discrepancy. PathComp's rule fixes two of these choices (Figure 1): *freeze the target*, computing $y_i$ once, and *cache the interface*, taking $u_i$ from cached inputs of later modules. Memory selection and confidence weights determine how these constraints are allocated.

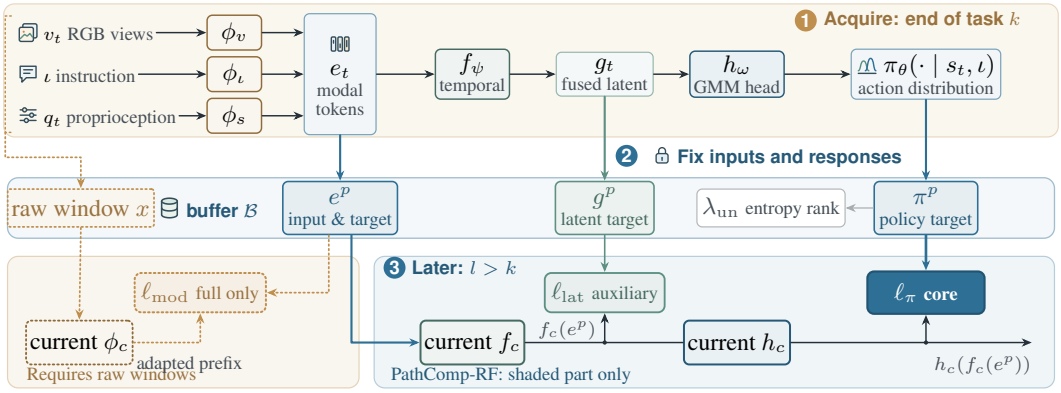


Figure 2: Persistent anchors along the VLA path. **Acquisition:** the adapted policy records $(e^p, g^p, \pi^p)$ after task $k$; full PathComp also stores the raw window $x$. **Later tasks:** the current fusion module and head act on $e^p$. The blue policy core compares $h_c(f_c(e^p))$ with $\pi^p$; the teal latent target $g^p$ supervises $f_c$ and is not an input to $h_c$. Only the dotted ochre branch re-encodes raw data. PathComp-RF retains the blue-shaded downstream branches and discards raw windows.

---

1: **for** $k = 1, \dots, K$ **do** — $\mathcal{B} \leftarrow \emptyset$ before the first task
2: **for** each training step **do**: $\mathcal{L}_{\text{BC}}^{(k)}$ on a minibatch of $\mathcal{D}^{(k)}$, in training mode
3: **if** $\mathcal{B} \neq \emptyset$: draw $\mathcal{P} \subset \mathcal{B}$; for each $b_i \in \mathcal{P}$, in evaluation mode:
4: $\ell_\pi^{(i)} \leftarrow \widehat{D}_{\text{KL}}\big(h_c(f_c(e_i^p)) \,\|\, \pi_i^p\big)$ on the cached tokens, against the fixed target — core
5: $\ell_{\text{lat}}^{(i)} \leftarrow \|f_c(e_i^p) - g_i^p\|_{\mathcal{G}}^2$ — [auxiliary]
6: $\ell_{\text{mod}}^{(i)} \leftarrow \|\phi_c(x_i) - e_i^p\|_{\mathcal{Z}}^2$ on the stored raw window — [full only]
7: update the trainable parameters with $\nabla\big(\mathcal{L}_{\text{BC}}^{(k)} + \lambda_\pi \langle \lambda_{\text{un}} \ell_\pi \rangle + \lambda_{\text{feat}} \langle \lambda_{\text{un}} (\ell_{\text{lat}} + \ell_{\text{mod}}) \rangle\big)$ — Eq. (6)
8: **end of task:** run the adapted model on $n_k$ windows of $\mathcal{D}^{(k)}$ in evaluation mode, no gradient
9: form anchor candidates with persistent targets $(e^p, g^p, \pi^p)$ — core
10: keep the raw window $x$ with each anchor — [full only]
11: set $\lambda_{\text{un}}$ by entropy rank, Eq. (5) — [heuristic; $\lambda_{\text{un}} = 1$ without it]
12: insert each candidate if space remains; at capacity, replace the highest-entropy anchor of a most-represented task only if the candidate has lower entropy; otherwise discard it

---

Algorithm 1: PathComp; labels distinguish the core, auxiliary and full-only terms. PathComp-RF skips lines 6 and 10; PathComp-fresh recomputes the targets of lines 4–6 with its own previous-task model.

### 3.2 Freeze the Target: Persistent Anchors

A target that a later model recomputes moves with that model, so PathComp computes each task's targets once, when the task is learned. At the end of task $k$, the adapted model, in evaluation mode, caches an *anchor* $b_i$ for each of $n_k$ windows of $\mathcal{D}^{(k)}$: its task, the modal tokens $e_i^p$, fused latents $g_i^p$ and GMM parameters $\pi_i^p$ of every step, and an entropy estimate $\widehat{\mathrm{H}}_i$. No action labels are stored; full PathComp also keeps the raw window $x_i = (s_i, \iota_i)$ for its modal anchor. These acquisition-time targets are *persistent*: they are never refreshed, and no teacher runs during later tasks. A rolling teacher (Li & Hoiem, 2018; Roy et al., 2025) instead re-targets task $k$ during task $l$ to $\pi_{l-1}$, the policy after task $l-1$, which carries the drift of the tasks in between and costs a teacher copy and a second forward pass. Persistence has its own cost: it also freezes the acquisition-time model's errors. The fresh-target control *PathComp-fresh* keeps the anchor windows, losses, weighting rule and cached-input pathway, but uses its own previous-task model's targets: $\phi_{l-1}(x_i)$, $f_{l-1}(e_i^p)$ and $h_{l-1}(f_{l-1}(e_i^p))$ for the modal, latent and policy terms during task $l$. Once the two training runs diverge, their cached tensors and entropy-derived weights need not remain numerically identical.

### 3.3 Cache the Interface: Evaluation on Cached Tokens

Cached modal tokens let the retention objective evaluate the current downstream modules on the inputs seen at acquisition, independently of changes in the encoders. Full PathComp constrains those encoder changes separately using stored raw windows; removing that constraint gives the raw-free variant. While a later task $l > k$ is learned, three terms compare the current maps $\phi_c, f_c, h_c$ with an anchor:

$$\ell_\pi^{(i)} = \frac{1}{T}\sum_t D_{\mathrm{KL}}\big(h_c(f_c(e_i^p))_t \,\big\|\, \pi_{i,t}^p\big), \quad \ell_{\mathrm{lat}}^{(i)} = \big\|f_c(e_i^p) - g_i^p\big\|_{\mathcal{G}}^2, \quad \ell_{\mathrm{mod}}^{(i)} = \big\|\phi_c(x_i) - e_i^p\big\|_{\mathcal{Z}}^2. \tag{3}$$

Here $\|\cdot\|_{\mathcal{G}}^2$ and $\|\cdot\|_{\mathcal{Z}}^2$ sum squared errors over the latent or tokens of a step and average over the $T = L_o$ steps. The **policy anchor** is the core term: it runs the cached tokens through the current fusion module and head (not the cached latent through the head) and compares the result with the cached action distribution. We estimate $\ell_\pi$ by reparameterized Monte Carlo, a partial pathwise estimator: the mixture component is drawn without a gradient path, so the mixture logits receive no expected gradient from this estimator (Graves, 2016). The anchor therefore acts directly through component means and scales; shared features and behavior cloning can still change the mixture weights. The **latent anchor** is auxiliary: it matches the fused latent on the same tokens, the state from which the head computes every mixture parameter, those weights included. The **modal anchor** re-encodes the stored raw window and is the only anchor term that observes encoder drift.

For the evaluation-mode maps, the path drift of one anchor splits into the parts the anchors see. With $\pi_c = h_c \circ f_c \circ \phi_c$, $\tilde{\pi}_c = h_c(f_c(e^p))$ and the step-averaged total variation $D(\pi, \pi') = \frac{1}{T}\sum_t \mathrm{TV}(\pi_t, \pi'_t)$, the triangle and Pinsker inequalities give

$$D(\pi_c(x), \pi^p) \le \underbrace{D\big(\pi_c(x), \tilde{\pi}_c\big)}_{\text{encoder drift}} + \min\Big\{\sqrt{\ell_\pi/2},\ \underbrace{D\big(\tilde{\pi}_c, h_c(g^p)\big)}_{\text{latent drift}} + \underbrace{D\big(h_c(g^p), \pi^p\big)}_{\text{head drift at } g^p}\Big\}. \tag{4}$$

Only the encoder drift involves raw inputs, and cached tokens do not expose it. It vanishes, and cached-token evaluation equals raw-input evaluation, when every module up to the cached interface is frozen ($\phi_c = \phi_k$) (Pellegrini et al., 2020); the modal term is then omitted. Full PathComp includes it because our encoders adapt; dropping it gives *raw-free PathComp (PathComp-RF)*, which stores no raw observation and leaves encoder drift unconstrained. For the exact KL, the policy anchor alone bounds the drift downstream of the cache at the anchor inputs, so the latent term is auxiliary (Appendix A.2).

### 3.4 Implementation Choices: Anchor Weights and Memory

Persistence keeps diffuse targets as they are, so PathComp down-weights them with a *confidence weight*, the entropy-rank weight

$$\lambda_{\mathrm{un}}^{(i)} \triangleq 1 - (1-\rho)\, r_i/(n_k - 1), \qquad \rho = 0.05, \tag{5}$$

where $r_i$ is the rank (0 for the lowest) of $\widehat{\mathsf{H}}_i$, a Monte Carlo estimate of the window-averaged differential entropy of $\pi_i^p$, among the $n_k$ candidates of task $k$; the weight is computed once and lies in $[\rho, 1]$. It is a heuristic: low entropy marks where the old policy was committed, not where it was correct. Each task inserts $n_k = \max(8, \lfloor B_{\max}/K \rfloor)$ candidates, drawn uniformly with replacement by a task-seeded generator, so all seeds and variants anchor the same windows. A full buffer replaces the highest-entropy anchor of a most-represented task only if the candidate has lower entropy, and otherwise discards the candidate (0 evictions in our streams; Algorithm 1, line 12).

### 3.5 Objective and Algorithm

With $\langle\cdot\rangle_{\mathcal{P}}$ the mean over an anchor minibatch $\mathcal{P} \subset \mathcal{B}$ of 8 anchors, PathComp uses the partial-gradient estimator above to optimize

$$\mathcal{L} = \underbrace{\mathcal{L}_{\mathrm{BC}}^{(l)} + \lambda_\pi \big\langle \lambda_{\mathrm{un}}^{(i)} \ell_\pi^{(i)} \big\rangle_{\mathcal{P}}}_{\text{core}} + \underbrace{\lambda_{\mathrm{feat}} \big\langle \lambda_{\mathrm{un}}^{(i)} \ell_{\mathrm{lat}}^{(i)} \big\rangle_{\mathcal{P}}}_{\text{auxiliary}} + \underbrace{\lambda_{\mathrm{feat}} \big\langle \lambda_{\mathrm{un}}^{(i)} \ell_{\mathrm{mod}}^{(i)} \big\rangle_{\mathcal{P}}}_{\text{full PathComp only}}, \tag{6}$$

with $\lambda_\pi = \lambda_{\mathrm{feat}} = 0.2$. The core combines behavior cloning and the policy anchor; confidence weights may be set to one. Full PathComp uses rank weights and adds both feature anchors; PathComp-RF drops only the modal anchor. PathComp adds no module at test time, so its inference cost equals that of the base policy; only full PathComp re-encodes stored windows during training.

# 4 EXPERIMENTS

## 4.1 PROTOCOL AND COMPARISONS

**Policy and streams.** We use the CLIP–LoRA–GMM policy of Section 3.1 on 10-task LIBERO streams (Liu et al., 2023), in alphabetical demonstration-file order. LIBERO-Object is the primary suite; Goal and Spatial provide cross-suite diagnostics. Every method trains 2000 steps per task with batch size 12 and a six-step context, using the same architecture, optimizer and schedule. PathComp's constants were selected during development on Object and then fixed; M2Distill-R uses published weights and DER++-style uses untuned defaults. The comparison therefore does not match hyperparameter-selection effort.

**Paired offline evaluation.** The first 40 of each task's 50 demonstrations train the policy; the remaining ten are held out. Every run uses the same 192 held-out windows per task, independent of seed, method and task order. The primary endpoint is squared action error, summed over seven action dimensions: $E_{l,k}$ averages $\|\hat{a}_w^{(l)} - a_w\|_2^2$ on task $k$ after training through task $l$. The decoded action $\hat{a}_w^{(l)}$ and demonstrated action $a_w$ come from the last step of each window $w$. We use *action MSE* as shorthand for the mean over windows of this summed squared error, rather than an average over action coordinates. We report

$$\text{Final} = \tfrac{1}{K}\textstyle\sum_k E_{K,k}, \quad \text{Forget} = \tfrac{1}{K-1}\textstyle\sum_{k<K} \tfrac{1}{K-k}\sum_{l>k}(E_{l,k} - E_{k,k}), \quad \text{Immed} = \tfrac{1}{K}\textstyle\sum_k E_{k,k}, \tag{7}$$

called *final error*, *forgetting* and *immediate error*; lower is better, and negative forgetting denotes backward transfer in prediction error. Decoding selects the mean of the component with the largest mixture-weighted component peak, using both mixture weights and scales. Our sensitivity endpoint replaces $E_{l,k}$ with behavior-cloning NLL $N_{l,k}$, averaged over all context steps and evaluated under the full mixture. MSE assesses a decoded action; NLL assesses the probability assigned to demonstrations. Neither is a LIBERO success rate.

**Baselines and resource matching.** *Sequential* uses behavior cloning alone. *M2Distill-R* reimplements M2Distill (Roy et al., 2025) with its published loss weights and a previous-task teacher at the modal and policy interfaces, without a fused-latent term. *DER++-style* replay (Buzzega et al., 2020) combines stored action labels with a reverse KL to stored GMM outputs, filling its buffer at task end rather than by reservoir sampling. All memory methods retain the same 20 windows per task and use the same KL estimator. This matches the stored examples, not bytes, replay batch size or compute: M2Distill-R merges 12 replay windows with the current batch and runs a teacher; PathComp uses eight anchors. This common-policy adaptation is not a reproduction of M2Distill's published architecture and success-rate protocol (Appendix D). Multimodal latent replay (MLR) (Yu et al., 2026), O-LoRA (Wang et al., 2023) and InfLoRA (Liang & Li, 2024) remain relevant untested alternatives.

**Seeds and uncertainty.** Object uses three training seeds; Goal and Spatial use two for each evaluated method. Seeds vary initialization and training draws, while anchor selection stays fixed by task. We report sample standard deviations, mean seed-paired differences and counts of favorable seeds; with three pairs, the smallest two-sided exact sign-test $p$-value is 0.25. Task-1 errors agree across methods at the displayed precision, as expected before any memory exists.

## 4.2 LOWER FINAL ACTION ERROR ON LIBERO-OBJECT

In Table 1, PathComp has the lowest final MSE, forgetting and immediate MSE among the evaluated continual methods, with lower final MSE in every paired seed. PathComp reaches 0.341 final MSE, compared with 0.440 for DER++-style replay and 0.461 for M2Distill-R. Relative to the strongest evaluated prior-work baseline under the shared policy and protocol, final MSE is reduced by 22.6%. The mean paired differences are $-0.099$ and $-0.120$, respectively, favoring PathComp in all three seeds. Against Sequential the difference is $-0.184$, again in all three seeds. These comparisons assess the complete methods: DER++-style also differs in label replay, constrained interfaces and training-mode evaluation, so it cannot isolate target timing or the benefit of caching.

Table 1: PathComp achieves the lowest final action error, forgetting and immediate error among the evaluated continual methods on LIBERO-Object. Final MSE is mean±sample standard deviation over three seeds; other metrics are seed means. Lower is better. ΔMSE is PathComp minus the row, with favorable seeds/pairs. Fresh and RF are PathComp variants; joint training is a one-seed offline reference. Percent reductions in the text use unrounded means.

| Method | Final MSE↓ | Final NLL↓ | Forget.↓ | Immed.↓ | ΔMSE (wins) |
|---|---|---|---|---|---|
| Sequential | 0.525±0.025 | −5.238 | 0.104 | 0.366 | −0.184 (3/3) |
| M2Distill-R | 0.461±0.020 | **−12.850** | 0.061 | 0.361 | −0.120 (3/3) |
| DER++-style | 0.440±0.026 | −10.993 | 0.047 | 0.362 | −0.099 (3/3) |
| **PathComp (ours)** | **0.341**±0.024 | −11.552 | **−0.015** | **0.353** | – |
| PathComp-fresh | 0.400±0.021 | −10.152 | 0.022 | 0.359 | −0.059 (3/3) |
| PathComp-RF | 0.426±0.014 | −9.298 | 0.041 | 0.358 | −0.086 (3/3) |
| Joint (reference) | 0.330 | −14.143 | – | – | – |

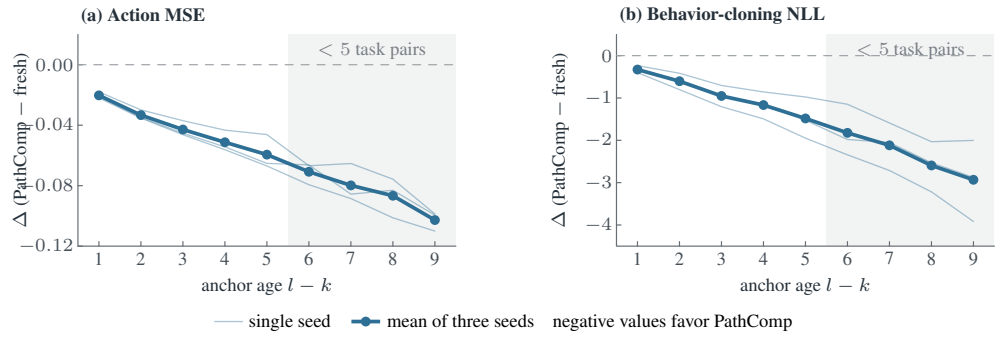


Figure 3: Target persistence across two endpoints on LIBERO-Object. Differences are PathComp minus fresh targets in **(a)** action MSE and **(b)** behavior-cloning NLL, averaged over the $10 - a$ task pairs at age $a = l - k$. Thin lines show individual seeds and thick lines their mean; lines join the age-wise averages. Shading marks fewer than five task pairs. Age varies with task identity, stream position and training trajectory; the plots are descriptive.

The lower final error is accompanied by negative mean forgetting, −0.015, without an evident loss of immediate fit. PathComp's immediate error is 0.353, compared with 0.366 for Sequential and 0.361 for M2Distill-R. Thus the retention improvement is not explained by simply failing to learn each new task.

### 4.3 Acquisition-Time Targets Improve Retention

PathComp-fresh uses the same anchor-window schedule, loss definitions, weighting rule and cached-input pathway, but replaces acquisition-time targets with targets from its own previous-task model. This adopts the rolling-target principle used by LifelongVLA (He et al., 2026) without reproducing that method in full. Final MSE rises from 0.341 to 0.400, with a paired difference of −0.059 favoring persistent targets in every seed. The control supports acquisition-time targets as an algorithmic choice under the shared retention design.

**Variation with anchor age.** For an anchor acquired at task $k$ and used during task $l$, age $a = l - k$ measures how many tasks separate acquisition from replay. At age one, acquisition-time and previous-task targets coincide *within each run*; this does not make targets identical across the two runs. At later ages, the fresh target is produced after $a - 1$ intervening updates. The fitted MSE difference has a negative age slope in every seed (mean −0.0097); NLL also has negative slopes in all seeds (mean −0.3099). Each slope fits the task-pair cells, equivalently weighting each age mean by $10 - a$. Figure 3 shows both endpoints: older tasks have larger error differences favoring persistent targets. Age also varies with task identity, position and training trajectory, so this pattern is consistent with accumulated target drift without identifying its causal contribution.

Table 2: Feature and policy constraints contribute to lower action error. Paired LIBERO-Object ablations (three seeds; mean±sample standard deviation; lower is better). Feature-only removes the policy term; policy-only removes both feature terms; RF removes only the modal term. Uniform weights change both allocation and mean regularization strength.

| Variant | Final MSE↓ | Forgetting↓ | Immediate MSE↓ |
|---|---|---|---|
| **Full PathComp** | **0.341**±0.024 | −**0.015**±0.005 | **0.353**±0.016 |
| Feature anchors only | 0.440±0.011 | 0.048±0.004 | 0.360±0.013 |
| Policy anchor only | 0.462±0.017 | 0.063±0.002 | 0.360±0.015 |
| PathComp-RF (no modal anchor) | 0.426±0.014 | 0.041±0.004 | 0.358±0.020 |
| Uniform anchor weights | 0.389±0.019 | 0.016±0.005 | 0.357±0.017 |
| Previous-task targets (fresh) | 0.400±0.021 | 0.022±0.005 | 0.359±0.021 |

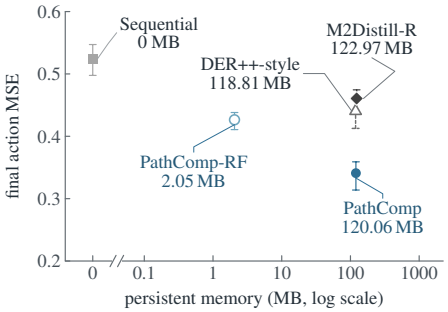


Figure 4: Storage and action error on LIBERO-Object. Markers show three-seed means; bars span the seed range. Memory is analytic with uint8 images; positive values use a log axis, with zero separated by a break. M2Distill-R counts trainable teacher parameters and buffers at 20 windows per task. Table 7 reports float32 storage and the full teacher copy.

### 4.4 Component Constraints and the Storage Trade-Off

**Feature and policy constraints.** Combining feature and policy constraints yields lower final error than either family alone (Table 2). Feature-only gives final MSE 0.440 and policy-only gives 0.462, compared with 0.341 for the full design, while immediate errors remain close. Policy-only removes both modal and latent terms, so this comparison supports their combined role but does not isolate the latent term. Uniform weights raise final MSE to 0.389. This tests the complete weighting rule: rank weights average $(1+\rho)/2$, whereas uniform weights equal one, changing both allocation and regularization strength.

**Raw observations constrain encoder drift.** PathComp-RF removes only the modal anchor and raw windows. Its final MSE is 0.426, higher than full PathComp in every paired seed but lower than Sequential (mean difference −0.098). With float32 images as stored, full PathComp uses 473.96 MB and PathComp-RF uses 2.05 MB after ten tasks. Full PathComp is therefore comparable in storage to DER++-style (472.71 MB) and M2Distill-R (476.87 MB, counting trainable teacher state). Figure 4 uses analytic uint8-image footprints; Table 7 also reports the float32 footprints and the full teacher copy used by the M2Distill-R implementation. The RF comparison measures the cost of removing the implemented encoder constraint, not an isolated benefit of cached evaluation. In the seed-0 profile, full PathComp takes 0.350 s per step, versus 0.300 for DER++-style, 0.360 for M2Distill-R and 0.310 for RF (Table 7). Eliminating the teacher therefore does not make the full method uniformly faster.

### 4.5 Cross-Suite Results and Metric Dependence

**NLL changes the method ordering.** M2Distill-R has lower final NLL than PathComp (−12.850 versus −11.552) in every Object seed (Table 9). PathComp still has lower final NLL than fresh targets and DER++-style. The MSE advantage thus does not imply higher likelihood under the full mixture.

**Goal and Spatial suites.** With its constants fixed after development on Object, PathComp has lower final action MSE than M2Distill-R and Sequential in both paired seeds on Goal and Spatial (Table 3). The reduction therefore extends to all three evaluated suites without per-suite selection of constants. These cross-suite comparisons use two seeds and evaluate Sequential, M2Distill-R and PathComp.

Table 3: Lower action error on additional LIBERO suites. Final held-out MSE (two seeds; mean±sample standard deviation). PathComp retains the constants selected on Object; DER++-style, RF and fresh targets were not evaluated here.

| Method | LIBERO-Goal↓ | LIBERO-Spatial↓ |
|---|---|---|
| Sequential | 0.581±0.031 | 0.554±0.002 |
| M2Distill-R | 0.505±0.014 | 0.470±0.017 |
| **PathComp** | **0.397**±0.019 | **0.366**±0.032 |

Table 4: Preserving the composed response permits adaptation within the head. LIBERO-Object checkpoint diagnostic (seed 0; mean over task pairs $k < l$ on fixed held-out windows). Modal and fusion drift are Euclidean norms; fusion holds the input at task-$k$ tokens, and head KL holds it at task-$k$ latents. Full-path KL evaluates the policy on raw inputs. KL uses the same Monte Carlo estimator as the policy loss. Lower values mean less drift.

| Method | Modal drift | Fusion drift \| $e$ | Head KL \| $g$ | Full-path KL |
|---|---|---|---|---|
| Sequential | 0.403 | 0.183 | 0.055 | 0.257 |
| M2Distill-R | 0.290 | 0.132 | 0.040 | 0.185 |
| PathComp-fresh | 0.250 | 0.114 | **0.034** | 0.159 |
| **PathComp** | **0.194** | **0.088** | 0.062 | **0.123** |

**Retention of the composed path.** The single-seed interface diagnostic tests how changes in individual modules relate to changes in their composition (Table 4; full results in Table 12). Compared with Sequential, PathComp reduces modal and fixed-input fusion drift, and lowers full-path policy KL from 0.257 to 0.123. Yet head KL on fixed cached latents is higher (0.062 versus 0.055). The fresh-target control has the lowest head KL on fixed latents but higher full-path KL than PathComp. This pattern is consistent with the implemented constraint: PathComp preserves $h_c(f_c(e^p))$, so the head can adapt together with the fusion module while retaining their composed response. In these single-seed measurements, isolated-module and composed-policy stability rank methods differently, which motivates evaluating retention at the composed-policy level alongside interface-specific drift.

**One-seed references.** A joint policy trained on all Object tasks for the same total 20,000 steps reaches 0.330 final MSE; it is an offline reference, not a continual baseline. With the task order reversed, PathComp reaches 0.368 final MSE versus 0.539 for Sequential (Table 11).

**OpenVLA-7B diagnostic.** A separate single-stage study with OpenVLA-7B (Kim et al., 2025) does not establish a closed-loop advantage. Its raw-input and label replay, its different KL and a task-index mismatch prevent a test of the central target-persistence claim (Appendix G).

## 5 Conclusion and Limitations

PathComp preserves acquisition-time interface inputs and responses while a shared policy continues to adapt. Downstream modules learn against fixed references, and a separate modal constraint addresses encoder changes. In offline evaluations on three LIBERO suites, PathComp lowers final action error relative to the compared continual baselines. On Object, the fresh-target control and component ablations support using acquisition-time targets and combining policy with feature constraints. These results suggest that retention can be organized around stable input–response relations along an adapting policy, with raw observations used where encoder changes must also be constrained.

**Limitations.** Our controlled evaluation uses one policy architecture and offline action prediction; closed-loop retention and transfer to other policy families, including action-pretrained VLAs, remain to be established. The MSE advantage over M2Distill-R does not carry over to held-out likelihood. Full PathComp stores raw observations, while the raw-free variant trades part of its retention benefit for lower storage. More seeds, task orders and matched tuning budgets would strengthen the comparison (Appendix I).

## Ethics Statement

This work studies continual fine-tuning of manipulation policies in benchmark environments. It involves no human subjects, no personal data and no physical robot. We use the LIBERO demonstrations (Liu et al., 2023) and OpenVLA-7B weights (Kim et al., 2025) under the terms of their public releases. The instruction-paraphrase set is generated from LIBERO instructions and contains no personal data (Appendix H). Full PathComp retains raw observation windows and PathComp-RF retains derived features; neither provides a privacy guarantee. Use on hardware requires safety validation beyond the evaluations reported here.

## Reproducibility Statement

Appendix C specifies the architecture, trainable parameters, hyperparameters and memory sampling; Appendix B defines the estimators, and Appendix D describes the baselines and their resource differences. Appendix F gives the demonstration split, shared evaluation windows, metrics and consistency checks. Training seeds are reported with each table, and Appendix A and the supplementary material provide the assumptions and proofs. Code and checkpoints will be made public after acceptance, together with configurations, evaluation-window identifiers and per-seed results. The separate OpenVLA diagnostic has implementation-version and task-index limitations documented in Appendix G; the code to be released is not the exact version behind every reported arm. Appendix H describes the instruction-paraphrase set.

## Use of Large Language Models

GPT-4o generated the 240 instruction paraphrases used only for the appendix check of the base OpenVLA-7B checkpoint (Appendix H). LLM-based writing assistance was used only for language polishing and grammatical revision.

## References

Ari S. Benjamin, David Rolnick, and Konrad Kording. Measuring and regularizing networks in function space. In *International Conference on Learning Representations*, 2019.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. $\pi_0$: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems*, 2025.

Matteo Boschini, Lorenzo Bonicelli, Pietro Buzzega, Angelo Porrello, and Simone Calderara. Class-incremental continual learning into the eXtended DER-verse. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(5):5497–5512, 2023.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pp. 2165–2183, 2023.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. In *Advances in Neural Information Processing Systems*, volume 33, pp. 15920–15930, 2020.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K. Dokania, Philip H. S. Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

Ziyang Chen, Shaoguang Wang, Weiyu Guo, Qianyi Cai, He Zhang, Pengteng Li, Yiren Zhao, and Yandong Guo. PHASER: Phase-aware and semantic experience replay for vision-language-action models. *arXiv preprint arXiv:2606.03598*, 2026.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3366–3385, 2022.

Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. PODNet: Pooled outputs distillation for small-tasks incremental learning. In *European Conference on Computer Vision*, pp. 86–102, 2020.

Maximilian Du, Zhanyi Sun, Chen Xu, Paarth Shah, Masha Itkina, and Shuran Song. Memory anchors for continual robot learning. *arXiv preprint arXiv:2608.26545*, 2026.

Alex Graves. Stochastic backpropagation through mixture density distributions. *arXiv preprint arXiv:1607.05690*, 2016.

Tyler L. Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. REMIND your neural network to prevent catastrophic forgetting. In *European Conference on Computer Vision*, pp. 466–483, 2020.

Yao He, Gan Sun, Wenqi Liang, Fazeng Li, and Yang Cong. Towards human-like physical intelligence: Lifelong vision-language-action learning for robotic manipulation. *arXiv preprint arXiv:2607.14852*, 2026.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Jiaheng Hu, Jay Shim, Chen Tang, Yoonchang Sung, Bo Liu, Peter Stone, and Roberto Martín-Martín. Simple recipe works: Vision-language-action models are natural continual learners with reinforcement learning. In *Reinforcement Learning Conference*, 2026.

Ahmet Iscen, Jeffrey Zhang, Svetlana Lazebnik, and Cordelia Schmid. Memory-efficient incremental learning through feature adaptation. In *European Conference on Computer Vision*, pp. 699–715, 2020.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Conference on Robot Learning*, pp. 2679–2713, 2025.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Daehee Lee, Minjong Yoo, Woo Kyung Kim, Wonje Choi, and Honguk Woo. Incremental learning of retrievable skills for efficient continual task adaptation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 17286–17312, 2024.

Yuheng Lei, Sitong Mao, Shunbo Zhou, Hongyuan Zhang, Xuelong Li, and Ping Luo. Dynamic mixture of progressive parameter-efficient expert library for lifelong robot learning. *Transactions on Machine Learning Research*, 2026.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2018.

Yan-Shuo Liang and Wu-Jun Li. InfLoRA: Interference-free low-rank adaptation for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 23638–23647, 2024.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 44776–44791, 2023.

Huihan Liu, Changyeon Kim, Bo Liu, Minghuan Liu, and Yuke Zhu. Pretrained vision-language-action models are surprisingly resistant to forgetting in continual learning. In *International Conference on Machine Learning*, 2026.

Zuxin Liu, Jesse Zhang, Kavosh Asadi, Yao Liu, Ding Zhao, Shoham Sabach, and Rasool Fakoor. TAIL: Task-specific adapters for imitation learning with large pretrained models. In *International Conference on Learning Representations*, 2024.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Robotics: Science and Systems*, 2024.

Pingbo Pan, Siddharth Swaroop, Alexander Immer, Runa Eschenhagen, Richard E. Turner, and Mohammad Emtiyaz Khan. Continual deep learning by functional regularisation of memorable past. In *Advances in Neural Information Processing Systems*, volume 33, pp. 4453–4464, 2020.

Lorenzo Pellegrini, Gabriele Graffieti, Vincenzo Lomonaco, and Davide Maltoni. Latent replay for real-time continual learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10203–10209, 2020.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pp. 8748–8763, 2021.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. iCaRL: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P. Lillicrap, and Greg Wayne. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Ralf Römer, Yi Zhang, Yuming Li, and Angela P. Schoellig. CLARE: Continual learning for vision-language-action models via autonomous adapter routing and expansion. *IEEE Robotics and Automation Letters*, 11(7):8331–8338, 2026.

Kaushik Roy, Akila Dissanayake, Brendan Tidd, and Peyman Moghadam. M2Distill: Multi-modal distillation for lifelong imitation learning. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1429–1435, 2025.

Kaushik Roy, Giovanni D'urso, Nicholas Lawrance, Brendan Tidd, and Peyman Moghadam. SPREAD: Subspace representation distillation for lifelong imitation learning. In *2026 IEEE International Conference on Robotics and Automation (ICRA)*, 2026.

Gobinda Saha, Isha Garg, and Kaushik Roy. Gradient projection memory for continual learning. In *International Conference on Learning Representations*, 2021.

Michalis K. Titsias, Jonathan Schwarz, Alexander G. de G. Matthews, Razvan Pascanu, and Yee Whye Teh. Functional regularisation for continual learning with Gaussian processes. In *International Conference on Learning Representations*, 2020.

Weikang Wan, Yifeng Zhu, Rutav Shah, and Yuke Zhu. LOTUS: Continual imitation learning for robot manipulation through unsupervised skill discovery. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 537–544, 2024.

Liyuan Wang, Jingyi Xie, Xingxing Zhang, Hang Su, and Jun Zhu. HiDe-PET: Continual learning via hierarchical decomposition of parameter-efficient tuning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(8):6687–6702, 2025.

Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. Orthogonal subspace learning for language model continual learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10658–10671, 2023.

Yuxuan Wu, Guangming Wang, Zhiheng Yang, Tianchen Deng, Maoqing Yao, Brian Sheil, and Hesheng Wang. Continually evolving skill knowledge in vision language action model. *arXiv preprint arXiv:2511.18085*, 2025.

Yushi Yang, Xiangli Nie, and Chang Liu. FAGR: Feature-action generative replay for robot lifelong imitation learning. *IEEE Robotics and Automation Letters*, 10(12):12907–12914, 2025.

Fanqi Yu, Matteo Tiezzi, Tommaso Apicella, Cigdem Beyan, and Vittorio Murino. Lifelong imitation learning with multimodal latent replay and incremental adjustment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6740–6749, 2026.

Lu Yu, Bartłomiej Twardowski, Xialei Liu, Luis Herranz, Kai Wang, Yongmei Cheng, Shangling Jui, and Joost van de Weijer. Semantic drift compensation for class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6982–6991, 2020.

Hongquan Zhang, Jingyu Gong, Zhizhong Zhang, Xin Tan, and Yuan Xie. T2S: Tokenized skill scaling for lifelong imitation learning. In *2026 IEEE International Conference on Robotics and Automation (ICRA)*, 2026.

Libang Zhao, Qixin Zeng, Hongyin Zhang, and Donglin Wang. Information-theoretic constraints for continual vision-language-action alignment. *arXiv preprint arXiv:2603.13335*, 2026.

Da-Wei Zhou, Hai-Long Sun, Jingyi Ning, Han-Jia Ye, and De-Chuan Zhan. Continual learning with pre-trained models: A survey. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 8363–8371, 2024.

Jiarun Zhu, Yijun Hong, Xiaoquan Sun, Zetian Xu, Qijun He, Haijier Chen, Zhiyong Wang, Mingqi Yuan, Wenjun Zeng, and Jiayu Chen. Can vision-language-action models learn from real-world data continually without forgetting? *arXiv preprint arXiv:2605.26820*, 2026.

# A Analysis of Persistent Interface Anchors

This appendix states the results used in the main text: the drift decomposition behind Eq. (4) and the gradient property of the implemented policy estimator. The supplementary material gives the complete analysis, including all proofs, a bound for confidence-weighted anchors and the training-mode case of the OpenVLA diagnostic.

## A.1 Setting

All maps are evaluated with dropout and image augmentation disabled, as in the CLIP–LoRA–GMM anchor computation. An anchor acquired after task $k$ stores $e^p = \phi_k(x)$, $g^p = f_k(e^p)$ and $\pi^p = h_k(g^p)$, which remain fixed while the current maps $\phi_c, f_c, h_c$ adapt. For sequences of $T$ distributions, $D(P, Q) = \frac{1}{T}\sum_t \mathrm{TV}(P_t, Q_t)$ with total variation $\mathrm{TV}(P_t, Q_t) = \frac{1}{2}\int |p_t - q_t|$. The policy loss of Eq. (3) is the population reverse KL $\ell_\pi = \frac{1}{T}\sum_t \mathrm{KL}(\tilde{\pi}_{c,t} \| \pi_t^p)$ with $\tilde{\pi}_c = h_c(f_c(e^p))$; training uses its Monte Carlo estimate (Appendix A.3). The evaluation decoder selects, at the last step, the mean of the component with the largest mixture-weighted density at its own mean, i.e. the largest $\alpha_m / \prod_a s_{m,a}$ for mixture weights $\alpha_m$ and diagonal scales $s_{m,a}$; the distances below concern the full predictive mixture, not this selected mean.

## A.2 Encoder and Downstream Drift

**Proposition 1** (Stage-wise decomposition)**.** *For an anchor $(x, e^p, g^p, \pi^p)$ and the current policy $\pi_c = h_c \circ f_c \circ \phi_c$, Eq. (4) holds. If every map up to the cached interface is unchanged ($\phi_c = \phi_k$), the encoder term vanishes and $D(\pi_c(x), \pi^p) \le \sqrt{\ell_\pi / 2}$.*

*Proof.* The triangle inequality through $\tilde{\pi}_c$ gives $D(\pi_c(x), \pi^p) \le D(\pi_c(x), \tilde{\pi}_c) + D(\tilde{\pi}_c, \pi^p)$. The second term is at most $D(\tilde{\pi}_c, h_c(g^p)) + D(h_c(g^p), \pi^p)$ by the triangle inequality through $h_c(g^p)$, and at most $\sqrt{\ell_\pi/2}$ by Pinsker's inequality at each step and Jensen's inequality over steps. Taking the smaller bound gives Eq. (4). If $\phi_c = \phi_k$, then $\phi_c(x) = e^p$ and $\pi_c(x) = \tilde{\pi}_c$. □

The policy term therefore constrains the composed downstream response on $e^p$, and only the modal term observes encoder changes, which cached-token evaluation cannot expose. The latent term constrains an intermediate response; the bound does not require it.

## A.3 Gradient of the Implemented Policy Estimator

At one step, let the current mixture be $P = \sum_m \alpha_m \mathcal{N}(\mu_m, \mathrm{diag}(s_m^2))$ with $\alpha = \mathrm{softmax}(\zeta)$, and let $Q$ be the cached mixture, with densities $p$, $q$ and component densities $p_m$. The implementation draws $m_n \sim \mathrm{Cat}(\alpha)$ without a gradient path, sets $a_n = \mu_{m_n} + s_{m_n} \odot \eta_n$ with $\eta_n \sim \mathcal{N}(0, I)$, and computes $\hat{\kappa} = \frac{1}{N}\sum_n [\log p(a_n) - \log q(a_n)]$.

**Lemma 1** (Partial pathwise gradient)**.** *Assume strictly positive mixture weights and diagonal scales, and let $\nabla$ be the back-propagated gradient with $(m_n, \eta_n)$ held fixed. Then*

$$\mathbb{E}[\hat{\kappa}] = \mathrm{KL}(P\|Q), \qquad \mathbb{E}[\nabla_\zeta \hat{\kappa}] = 0, \qquad \mathbb{E}[\nabla_{\mu,s} \hat{\kappa}] = \nabla_{\mu,s} \mathrm{KL}(P\|Q).$$

*Proof sketch.* The construction gives $a_n \sim P$, so the value is unbiased. Without a gradient through the index, $\partial_{\zeta_m} \log p(a) = r_m(a) - \alpha_m$ with $r_m(a) = \alpha_m p_m(a)/p(a)$, and $\mathbb{E}_{a\sim P} r_m(a) = \alpha_m$. For means and scales, the law of $(m_n, \eta_n)$ does not depend on $(\mu, s)$, so the gradient passes under the expectation; the supplementary material states the regularity conditions. □

The anchor term therefore trains component means and scales directly but supplies no expected update through the mixture logits (Graves, 2016); the logits can still change through shared features and behavior cloning. Proposition 1 concerns the population KL, whereas a finite Monte Carlo estimate may be negative. The paired baselines with a policy term use the same estimator.

### A.4 Target Source and Scope

**Remark 1** (Target source)**.** On a fixed set of task-$k$ anchors with probability weights $\nu_i$, let $F_j = h_j \circ f_j$ and $\rho(F, F') = \sum_i \nu_i D(F(e_i^p), F'(e_i^p))$. If each previous-task target is fitted to within $\rho(F_j, F_{j-1}) \le \epsilon_j$, the triangle inequality gives only $\rho(F_l, F_k) \le \sum_{j=k+1}^{l} \epsilon_j$, whereas fitting the acquisition-time target to within $\epsilon_l$ gives $\rho(F_l, F_k) \le \epsilon_l$ directly.

The two fit conditions concern different targets, so the remark explains why a fixed reference removes a teacher-drift term but does not establish a performance ranking; Section 4.3 compares the training rules empirically.

The bounds concern predictive distributions on stored anchors in evaluation mode. They do not cover held-out states, the component-selecting decoder or closed-loop success, and no fit residual is estimated in the experiments. The OpenVLA instantiation recomputes its anchor terms with LoRA dropout active; by the convexity of KL and of the squared norm, its losses, in expectation over dropout masks, upper-bound the discrepancies of dropout-averaged predictions and latents, which need not equal the evaluation-mode outputs used in rollouts.

## B Estimators and Confidence Weights

We specify how the cached entropy, the confidence weights and the policy term are computed, and note implemented alternatives that no paired run uses.

**Entropy of a cached window.** At the end of task $k$, the adapted policy in evaluation mode outputs a Gaussian mixture $\pi_t$ at each step $t$ of a candidate window $\tau$. We estimate the differential entropy of each step by Monte Carlo and average it over the window:

$$\widehat{\mathsf{H}}(\tau) = \frac{1}{T}\sum_{t=1}^{T}\Big(-\frac{1}{N}\sum_{n=1}^{N}\log \pi_t\big(a_t^{(n)}\big)\Big), \qquad a_t^{(n)} \sim \pi_t, \tag{8}$$

with $N = 20$ samples and $T$ the context length. Differential entropy can be negative and depends on the action units; the rank form below uses only its order within a task.

**Confidence weights.** Except for the uniform-weight ablation, paired LIBERO PathComp runs use the rank form of Eq. (5); the ablation sets every weight to one. Within the candidates of one task, the most confident window gets weight 1, the least confident gets the floor $\rho = 0.05$, and the weights are linear in the entropy rank in between. The code also implements the absolute form $\lambda_{\mathrm{un}} = \min(\exp(-\gamma\widehat{\mathsf{H}}), 1)$. It equals 1 whenever the entropy estimate is nonpositive, so it assigns the same weight to all such windows. The OpenVLA instantiation uses the absolute form with $\gamma = 0.5$ on the mean Shannon entropy of its action-token distributions, which is nonnegative, so the cap is inactive there. Weights are computed once, when an anchor is cached, and are never re-scored.

**Policy term.** The policy term is the reverse divergence $D_{\mathrm{KL}}(\tilde{\pi}_{c,t} \,\|\, \pi_t^p)$, current policy first, averaged over the steps of the window. We estimate it by Monte Carlo with $N = 20$ samples from the current mixture. The sampler draws the component index without a gradient path and the Gaussian noise by reparameterization. The estimate is therefore unbiased in value and gives unbiased gradients for the component means and scales. Its expected gradient with respect to the mixture logits, however, is zero (Lemma 1), a known limit of reparameterizing mixtures (Graves, 2016); the logits can still change through shared features and the behavior-cloning objective. The policy term of M2Distill-R and the distribution term of DER++-style replay use the same estimator, direction and sample count, so this caveat applies to every method with a policy term. The code also contains a matched-component analytic bound and a forward-KL estimator, both of which pass gradients to the logits; no paired run uses them.

**KL direction.** The reverse direction is mode-seeking: the current policy may concentrate on some modes of a multimodal cached target instead of covering all of them. We use it because its samples come from the current policy, which lets the estimator pass gradients through the current head. A single-seed pilot scored by the legacy unpaired evaluator (Appendix F) could not separate reverse, forward and symmetric KL, so we make no claim about which direction is best.

**Evaluation mode for the anchor terms.** Anchors are cached by the adapted model in evaluation mode and without gradients. During later tasks, the anchor terms are recomputed with the current model also in evaluation mode, so dropout and image augmentation do not perturb them; gradients still flow. The behavior-cloning term is computed in training mode. OpenVLA differs: its anchor terms are computed with LoRA dropout active, so they constrain dropout-averaged predictions and latents (Appendix A.4).

## C Implementation Details

Each run records its full configuration; the values below are taken from these records and are collected in Table 5.

**Policy.** Each step observes two $128 \times 128$ RGB views (agent view and wrist camera), the 9-dimensional robot state recorded in the LIBERO demonstrations (Liu et al., 2023), and the task instruction. Each view is resized to $224 \times 224$ and encoded by the frozen CLIP ViT-B/16 visual tower (Radford et al., 2021). The MLP projections of every block of that tower carry LoRA adapters (Hu et al., 2022) of rank 8 with LoRA alpha 32, and a linear map projects the tower's output to the token width of 64. The instruction is embedded once by the frozen CLIP text encoder, and a two-layer MLP projects the 512-dimensional embedding to one token. A three-layer MLP maps the robot state to one token per step. The temporal module $f_\psi$ is a transformer with 4 layers, 4 heads, MLP width 256 and dropout 0.1. It attends causally over the tokens of the last 6 steps: the instruction, the state and the two views at each step. The head $h_\omega$ is a two-layer MLP of width 256 that outputs, at every step, a 5-component Gaussian mixture over the 7-dimensional action, with tanh-bounded means and softplus scales floored at $\sigma_{\min} = 10^{-4}$. The CLIP towers provide pretrained vision–language features; the projection MLPs, temporal module and GMM head are newly initialized, without an action-policy checkpoint before task one. LoRA initializes its first low-rank factor randomly and its second to zero. Actions are read from the demonstration files as seven coordinates (translation, rotation and gripper), cast to float32, and passed to the loss without further rescaling, normalization, clipping or gripper recoding. MSE therefore sums error in the stored action coordinates, rather than in a common physical unit. Training minimizes the negative log-likelihood of the demonstrated actions. For evaluation, the policy outputs the mean of the component with the highest mixture-weighted density at its own mean, at the last step of the window (Appendix A.1).

**Trainable parameters.** The LoRA adapters of the visual tower, the visual projection, the text projector, the state encoder, the temporal module and the head are trained: 1,117,707 parameters in 118 tensors. The pretrained CLIP weights stay frozen. Every map up to the cached modal tokens therefore adapts, which is why cached tokens cannot see encoder drift (Section 3.3).

**Optimization and data.** All methods use AdamW with learning rate $10^{-4}$, weight decay $10^{-4}$, gradient-norm clipping at 1, batch size 12 and 2000 steps per task, on streams of 10 tasks. Of the 50 demonstrations of a task, the first 40 in index order are used for training and the rest are held out. Windows of 6 steps are taken with stride 4, and a split with more than 2000 windows is capped by a permutation seeded by the task. Images are augmented by random translations (padding of 8 pixels) during training only.

**Anchors.** An anchor stores its task index, confidence weight, entropy and instruction embedding. It also stores the per-step vision and state tokens, the instruction token, and the fused latents and mixture parameters of all steps. Full PathComp also stores the raw window (both views and the robot state of each step), which only the modal term reads. No anchor stores action labels. The cached part takes 10,272 bytes per anchor and the raw window 2,359,512 bytes (float32 images).

**Other VLAs.** The three interfaces map onto other VLAs as follows. Modal anchors sit on the outputs of the vision and state encoders. Latent anchors sit on the language-model hidden state at the action positions (OpenVLA) or on the input of the action expert ($\pi_0$-style). For token-based VLAs, policy anchors sit on the action-token distribution, with token entropy in place of GMM entropy. Flow or diffusion heads would need a distribution-level surrogate for $\ell_\pi$, which we do not implement; this mapping is a design note, not a result.

Table 5: Configuration of the paired LIBERO runs, from the recorded run configurations. Every memory method keeps 20 windows per task. Flags are those of the training script we will release.

| Setting | Value |
|---|---|
| *All methods* | |
| Stream | 10 tasks in alphabetical file order; 2000 steps per task; batch 12; context 6 |
| Optimizer | AdamW; learning rate $10^{-4}$; weight decay $10^{-4}$; gradient-norm clip 1 |
| Evaluation | 192 fixed held-out windows per task (Appendix F) |
| KL estimator | reverse; reparameterized Monte Carlo; 20 samples |
| *PathComp*: `--reliability-mode rank --lambda-feat 0.2 --lambda-pi 0.2 --kl-estimator reparam` | |
| Buffer | $B_{\max} = 200$; 20 anchors per task; no eviction in a 10-task stream |
| Anchor minibatch | $\lvert\mathcal{P}\rvert = 8$, uniform without replacement |
| Loss weights | $\lambda_{\text{feat}} = \lambda_{\pi} = 0.2$ |
| Confidence weight | rank form, floor $\rho = 0.05$; entropy from 20 samples per step |
| Variants | PathComp-RF: `--no-modal`; PathComp-fresh: `--fresh-targets` |
| M2Distill-R | `--m2-official --n-memories 20`; all weights 0.05 on Object and Spatial; image and policy 0.25, text and state 0.05 on Goal |
| DER++-style | `--derpp-kl --n-memories 20`; label-replay and distribution weights 0.5 each |
| Joint | `--joint`; one policy on all tasks at once for 20,000 steps |

Table 6: Where continual-learning regularizers obtain their targets. The columns are the terms of Eq. (2), defined in Section 3.1; Figure 1 shows the target and input columns. Entries describe the published methods; this appendix describes the baselines as we implement them.

| Method | Target from | Where applied | Input at replay | Labels | Teacher |
|---|---|---|---|---|---|
| ER | labels | output | raw | yes | no |
| LwF | previous model | output | new-task data | yes | yes |
| iCaRL | previous model | output | raw | yes | yes |
| PODNet/M2Distill/SPREAD | previous model | features, output | raw | yes | yes |
| DER, DER++ | model at storage | output | raw | DER++ only | no |
| Latent replay, MLR | labels | above the cache | cached features | yes | no |
| LifelongVLA | labels, prev. model | above the cache | cached tokens | yes | yes |
| LOTUS | labels | output | raw | yes | no |
| PathComp | model at acquisition | policy, latent, modal | cached; raw (modal) | no | no |
| PathComp-RF | model at acquisition | policy, latent | cached | no | no |

**How the constants were set.** PathComp's constants ($\lambda_{\text{feat}} = \lambda_{\pi} = 0.2$, the floor $\rho = 0.05$, $|\mathcal{P}| = 8$, $B_{\max} = 200$) were fixed during development, and no paired run changed them. They were chosen on LIBERO-Object with the legacy unpaired evaluator, never on LIBERO-Goal or LIBERO-Spatial. The baselines use published weights (M2Distill) or untuned defaults (DER++-style, both weights 0.5).

**Anchor and memory windows.** At the end of each task, the windows to cache are drawn uniformly with replacement from the task's training windows. The generator is seeded by the task and the number of windows, not by the training seed. Every seed, every PathComp variant and every memory baseline therefore stores the same 20 windows of each task, and seed-to-seed variation does not cover anchor selection. The anchor minibatch is drawn uniformly without replacement at every step.

**Compute.** Each run uses one GPU. Step timing includes data loading, excludes task-end anchor construction and the first ten warm-up steps of each task, and is averaged over tasks after the first. The recorded profiles do not identify the GPU model, so we use these timings descriptively rather than as hardware-normalized comparisons. Table 7 reports measured time per step and peak memory from the seed-0 runs.

## D Baselines as Implemented

Table 6 compares where PathComp and published regularizers obtain their targets. All baselines use the same training code, policy, optimizer, data, evaluation windows and KL estimator as PathComp. At the first task every memory is empty, and every method updates exactly as Sequential does; Appendix F describes the regression test that checks this.

**Sequential.** Behavior cloning on the current task only.

**M2Distill-R.** M2Distill-R is our adaptation of M2Distill (Roy et al., 2025) to the shared CLIP–LoRA–GMM policy, with the published loss weights and 20 memory windows per task. Its architecture, memory protocol and offline metric differ from the published evaluation. After each task, a frozen copy of the policy becomes the teacher for the next task. Each step merges the current batch with an equally large batch of replayed windows, and applies behavior cloning and distillation to the merged batch. Distillation has four terms: squared $\ell_2$ distances between student and teacher tokens for the images (summed over the two views), the instruction and the robot state, and the reverse KL between their action distributions. There is no fused-latent term. The weights are the published ones: 0.05 for every term on LIBERO-Object and -Spatial; on LIBERO-Goal, 0.25 for the image and policy terms and 0.05 for the text and state terms. The teacher sees exactly the student's augmented pixels; the student runs in training mode and the teacher in evaluation mode. Matching the item count to PathComp does not match batch composition or compute: M2Distill-R distills on a merged batch twice the current one and runs a teacher pass, while PathComp adds an anchor minibatch of $|\mathcal{P}| = 8$. A teacher needs only the trainable parameters and buffers, 4,522,052 bytes; our implementation deep-copies the whole policy, including the frozen CLIP tower shared with the student (349,292,612 bytes).

**DER++-style replay.** Each step adds a replay minibatch, as large as the current batch while memory allows. Two terms act on it, each with weight 0.5: behavior cloning on the stored action labels, and the reverse KL from the current mixture to the mixture stored for that window. The memory keeps 20 windows per task, filled at the end of each task with the outputs of the end-of-task model in evaluation mode. The student computes both replay terms in training mode, with augmentation and dropout, against targets cached in evaluation mode; PathComp's anchor terms run in evaluation mode (Appendix B). This differs from DER++ (Buzzega et al., 2020) in two ways. First, the stored target is compared at the level of distributions rather than by a squared error on output parameters, because mixture parameters are not invariant to the order of the components. Second, the memory is filled at task end rather than by reservoir sampling.

**PathComp-fresh.** The fresh-target control uses the same anchor-window schedule, loss definitions, weighting rule and cached-input pathway as PathComp. Cached tensors and weights depend on each run's own training trajectory. At the end of each task it stores a frozen copy of the model, and during task $l$ this copy recomputes every target: $\phi_{l-1}(x)$ for the modal term, $f_{l-1}(e^p)$ for the latent term and $h_{l-1}(f_{l-1}(e^p))$ for the policy term. For anchors cached at the end of task $l-1$, this copy is the model that cached them: within this run, acquisition-time and previous-task targets coincide at age one. Targets need not coincide across runs. Like M2Distill-R, the control keeps a frozen model copy.

**PathComp-RF.** The raw-free variant drops the modal term and stores no raw window; everything else is unchanged.

**Joint reference.** One policy is trained by behavior cloning on the union of all ten tasks' training windows, sampled in proportion to their numbers, for 20,000 steps at batch 12. It is evaluated once, on the same windows as the continual runs, so only its final error is defined.

**Methods not run.** Our implementation also includes EWC (Kirkpatrick et al., 2017) and ER (Chaudhry et al., 2019). Neither was run under the paired protocol, and we report no number for them; DER++-style replay contains ER's label replay as one of its terms. Isolation and skill-growth methods such as TAIL (Liu et al., 2024) and LOTUS (Wan et al., 2024) add task-specific modules and need task identity or a skill router at test time. Their published LIBERO results use success metrics we do not measure, so we neither run them nor import their numbers. Three relevant alternatives were not run. MLR (Yu et al., 2026), latent replay with labels on frozen encoders, is the closest raw-free baseline. O-LoRA (Wang et al., 2023) and InfLoRA (Liang & Li, 2024) store no replay data and restrict each task's LoRA update to a subspace that interferes little with earlier tasks. Our comparisons therefore do not place PathComp against raw-free latent replay or rehearsal-free LoRA methods, and we make no claim beyond the baselines above. Other parameter-space methods (Kirkpatrick et al., 2017; Saha et al., 2021) likewise constrain the weights or their update directions rather than the outputs of old tasks.

Table 7: Persistent method-specific storage after ten tasks and seed-0 Object compute profiles. Analytic footprints use tensor shapes with float32 or uint8 images; MB denotes decimal megabytes. The Stored MB column uses the same accounting convention, excluding shared base weights; it is not total physical allocation. In particular, M2Distill-R counts 4.52 MB of trainable teacher state, while its implementation copies the frozen tower too and retains 821.64 MB. Seconds per step exclude the first task; peak memory includes training allocations. Full PathComp has replay-scale storage; only PathComp-RF keeps a memory without raw windows.

| | | | Analytic MB | | Measured profile | | |
|---|---|---|---|---|---|---|---|
| Method | Persistent state | Raw obs.? | fp32 img. | uint8 img. | Stored MB | s/step | Peak GB |
| Sequential | nothing | no | 0 | 0 | 0.00 | 0.225 | 6.1 |
| DER++-style | 20 raw windows/task + stored GMM outputs | yes | 472.71 | 118.81 | 472.71 | 0.300 | 7.8 |
| M2Distill-R | teacher copy (trainable) + 20 raw windows/task | yes | 476.87 | 122.97 | 476.87 | 0.360 | 9.4 |
| PathComp | 200 anchors incl. raw window | yes | 473.96 | 120.06 | 473.96 | 0.350 | 8.1 |
| PathComp-RF | 200 token/latent/GMM anchors | no | 2.05 | 2.05 | 2.05 | 0.310 | 6.7 |

**Storage accounting.** Table 7 counts the persistent method-specific state after ten tasks from the tensor shapes in the code, with images in float32 as stored. Let $N = 200$ be the number of stored items. An anchor's cached part takes $b_{\text{feat}} = 10{,}272$ bytes and its raw window $b_{\text{obs}} = 2{,}359{,}512$ bytes. A replayed window with its action labels and instruction embedding takes $b_{\text{seg}} = 2{,}361{,}736$ bytes, and the trainable part of the M2Distill-R teacher takes $b_{\text{T}} = 4{,}522{,}052$ bytes. Full PathComp then stores $N(b_{\text{feat}} + b_{\text{obs}})$ bytes and PathComp-RF stores $Nb_{\text{feat}}$. M2Distill-R stores $b_{\text{T}} + Nb_{\text{seg}}$, and DER++-style replay stores $Nb_{\text{seg}}$ plus the mixture parameters of each window. Model weights shared by all methods, including the frozen CLIP tower that M2Distill-R also copies into its teacher, are not counted; with that copy M2Distill-R keeps 821.64 MB.

## E Additional Results

This appendix reports cross-suite results, likelihood metrics, component ablations and interface diagnostics that complement the primary Object comparison.

**All suites.** Table 8 lists final error, forgetting and immediate error for every method and suite of the paired protocol. LIBERO-Goal and -Spatial use two training seeds for Sequential, M2Distill-R and PathComp, as diagnostics rather than primary evidence (Section 4.1). M2Distill-R uses the published Goal weights on LIBERO-Goal (Appendix D).

**Sensitivity endpoint: held-out NLL.** Table 9 reports the behavior-cloning negative log-likelihood of the demonstrated action under the full mixture, logged for every window alongside the squared error of the decoded action. NLL averages the negative log density of demonstrated actions over all context steps, whereas MSE scores the decoded action at the last step. Both depend on mixture weights and scales, but they measure different aspects of the prediction. The NLL ranking alone cannot identify mixture-weight collapse or explain its cause. Table 10 lists the per-seed paired differences behind the win counts of Tables 1 and 9.

**Variants and references.** Table 11 reports the three-seed component and target-time comparisons, with one-seed reversed-order and joint-training references. The uniform-weight comparison changes both weight allocation and mean strength: the rank weights average $(1+\rho)/2$, while uniform weights equal one. No row removes only the latent term from the full model. Reversing the order changes each task's position in the stream, so only aggregate metrics are comparable across orders.

**Per-interface drift.** Table 12 uses the end-of-task checkpoints of training seed 0. For each pair $k < l$, it compares the models after tasks $k$ and $l$ on fixed held-out windows of task $k$, and averages over all pairs. Three columns isolate one interface each. They measure the change of the modal tokens, the change of the fused latent when its input is held at task $k$'s tokens, and the reverse KL of the head when its input is held at task $k$'s latent. The other columns measure end-to-end changes of the latent, the action distribution and the decoded action. The reference outputs are checked by evaluating each reference model again on the same windows. Under Sequential, the representations

Table 8: Offline retention on 10-task LIBERO streams with the CLIP–LoRA–GMM policy. Final action MSE, forgetting and immediate error follow Eq. (7), with squared errors summed over seven action dimensions; lower is better, and negative forgetting denotes backward transfer. Cells are mean±sample standard deviation over training seeds; superscripts mark the two-seed Goal and Spatial diagnostics. All methods share the evaluation windows; memory methods retain 20 windows per task. M2Distill-R and DER++-style are the adaptations described in Appendix D. Dashes indicate methods not evaluated on a suite.

| | LIBERO-Object | | | LIBERO-Goal | | | LIBERO-Spatial | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | Final↓ | Forget.↓ | Immed.↓ | Final↓ | Forget.↓ | Immed.↓ | Final↓ | Forget.↓ | Immed.↓ |
| Sequential | 0.525 ±0.025 | 0.104 ±0.007 | 0.366 ±0.017 | 0.581 ±0.031 $^{n=2}$ | 0.108 ±0.010 $^{n=2}$ | 0.416 ±0.022 $^{n=2}$ | 0.554 ±0.002 $^{n=2}$ | 0.108 ±0.009 $^{n=2}$ | 0.390 ±0.025 $^{n=2}$ |
| DER++-style | 0.440 ±0.026 | 0.047 ±0.007 | 0.362 ±0.022 | – | – | – | – | – | – |
| M2Distill-R | 0.461 ±0.020 | 0.061 ±0.004 | 0.361 ±0.023 | 0.505 ±0.014 $^{n=2}$ | 0.058 ±0.002 $^{n=2}$ | 0.411 ±0.025 $^{n=2}$ | 0.470 ±0.017 $^{n=2}$ | 0.052 ±0.000 $^{n=2}$ | 0.385 ±0.029 $^{n=2}$ |
| PathComp | 0.341 ±0.024 | −0.015 ±0.005 | 0.353 ±0.016 | 0.397 ±0.019 $^{n=2}$ | −0.013 ±0.001 $^{n=2}$ | 0.403 ±0.018 $^{n=2}$ | 0.366 ±0.032 $^{n=2}$ | −0.016 ±0.011 $^{n=2}$ | 0.377 ±0.021 $^{n=2}$ |
| PathComp-RF | 0.426 ±0.014 | 0.041 ±0.004 | 0.358 ±0.020 | – | – | – | – | – | – |

Table 9: Held-out NLL on LIBERO-Object from the same runs and windows as Table 1, averaged over all context steps under the full Gaussian mixture; lower is better. Cells are mean±sample standard deviation over three seeds, except the one-seed joint reference. Final, forgetting and immediate NLL use Eq. (7); the last column gives PathComp minus the row in final NLL (wins/pairs). M2Distill-R has lower final NLL than PathComp in every seed, reversing their MSE ordering.

| Method | Final NLL↓ | Forget. NLL↓ | Immed. NLL↓ | ΔFinal NLL |
|---|---|---|---|---|
| Sequential | −5.238 ±0.217 | 4.826 ±0.126 | −12.622 ±0.425 | −6.314 (3/3) |
| M2Distill-R | −12.850 ±0.199 | −0.450 ±0.107 | −12.697 ±0.516 | +1.298 (0/3) |
| DER++-style | −10.993 ±0.460 | 0.872 ±0.063 | −12.687 ±0.446 | −0.559 (3/3) |
| PathComp-fresh | −10.152 ±0.443 | 1.477 ±0.153 | −12.707 ±0.387 | −1.400 (3/3) |
| PathComp-RF | −9.298 ±0.435 | 2.072 ±0.127 | −12.707 ±0.401 | −2.254 (3/3) |
| PathComp (ours) | −11.552 ±0.324 | 0.584 ±0.114 | −12.747 ±0.359 | – |
| Joint (reference) | −14.143 $^{n=1}$ | – | – | – |

of old-task windows drift at all three interfaces: the modal tokens, the fused latent given fixed tokens, and the head given a fixed latent (seed 0). Relative to Sequential, PathComp reduces the drift of old-task windows in the modal tokens, in the fused latent given fixed tokens and in the end-to-end action distribution (seed 0). No loss constrains the head at the cached latent (Eq. (4)), so we make no prediction for its drift (0.062 for PathComp, 0.055 for Sequential).

**Plasticity and forgetting.** Figure 5 shows the mean error increase of Object tasks by age, and Figure 6 places each method's seed mean by immediate error and forgetting, which shows whether a method trades plasticity for stability.

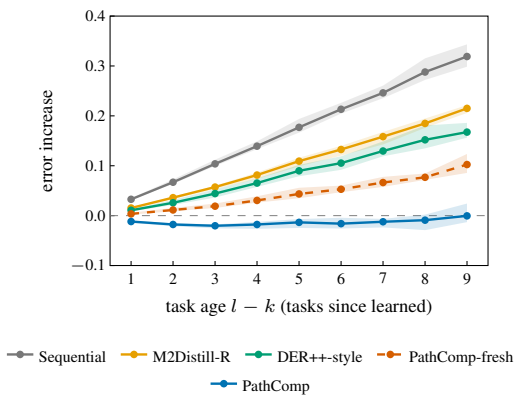


Figure 5: Per-age error increase on LIBERO-Object (offline held-out action error, paired evaluation, lower is better): mean increase of a task's error $l - k$ tasks after it was learned. Age $a$ averages the $10 - a$ tasks old enough, so the rightmost points rest on one or two tasks; lines are seed means, shaded bands the range over seeds (seeds: Sequential 3, M2Distill-R 3, DER++-style 3, PathComp-fresh 3, PathComp 3).

Table 10: Per-seed paired differences on LIBERO-Object (training seeds 0–2): PathComp minus the row in final action error (left) and in final held-out NLL (right), one column per training seed, from the runs and windows of Tables 1 and 9; negative: PathComp lower. The wins/pairs counts of those tables are the signs of these values. With three seeds we report the values themselves rather than bootstrap intervals.

| | ΔFinal (squared error) | | | ΔFinal NLL | | |
|---|---|---|---|---|---|---|
| Row | seed 0 | seed 1 | seed 2 | seed 0 | seed 1 | seed 2 |
| Sequential | −0.197 | −0.184 | −0.170 | −6.440 | −6.359 | −6.142 |
| M2Distill-R | −0.120 | −0.123 | −0.115 | +1.127 | +1.229 | +1.538 |
| DER++-style | −0.096 | −0.099 | −0.104 | −0.425 | −0.826 | −0.425 |
| PathComp-fresh | −0.055 | −0.063 | −0.059 | −1.428 | −1.770 | −1.003 |
| PathComp-RF | −0.081 | −0.097 | −0.079 | −2.204 | −2.572 | −1.985 |

Table 11: LIBERO-Object ablations under paired offline evaluation; lower is better. Component and target-time rows use three seeds (mean±sample standard deviation); superscripts mark one-seed references. Feature-only removes the policy loss; policy-only removes both feature losses; RF removes only the modal loss. Uniform weights change both allocation and average regularization strength. Fresh targets use the same window schedule, loss definitions, weighting rule and cached-input pathway, with targets from the run's own previous-task model. Reversed-order results are comparable only in aggregate; joint training has only a final error.

| Variant | Final↓ | Forget.↓ | Immed.↓ |
|---|---|---|---|
| Full PathComp | 0.341 ±0.024 | −0.015 ±0.005 | 0.353 ±0.016 |
| Feature anchors only ($\lambda_\pi = 0$) | 0.440 ±0.011 | 0.048 ±0.004 | 0.360 ±0.013 |
| Policy anchor only ($\lambda_{\text{feat}} = 0$) | 0.462 ±0.017 | 0.063 ±0.002 | 0.360 ±0.015 |
| PathComp-RF (no modal anchor) | 0.426 ±0.014 | 0.041 ±0.004 | 0.358 ±0.020 |
| Uniform anchor weights | 0.389 ±0.019 | 0.016 ±0.005 | 0.357 ±0.017 |
| PathComp-fresh (previous-task targets) | 0.400 ±0.021 | 0.022 ±0.005 | 0.359 ±0.021 |
| Reversed task order | 0.368 $^{n=1}$ | −0.004 $^{n=1}$ | 0.364 $^{n=1}$ |
| *Reference* | | | |
| Sequential, reversed task order | 0.539 $^{n=1}$ | 0.108 $^{n=1}$ | 0.376 $^{n=1}$ |
| Joint multitask reference | 0.330 $^{n=1}$ | – | – |

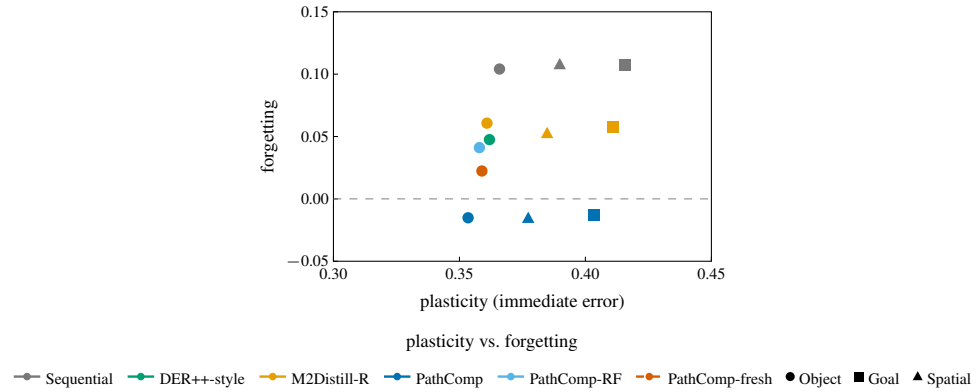


Figure 6: Immediate action error against forgetting (offline held-out action error, paired evaluation), per method and suite; markers are seed means; lower-left is better.

## F Evaluation Protocol

This appendix specifies data selection, metric aggregation and implementation checks for the paired evaluation.

**Split.** Each LIBERO task has 50 demonstrations. They are sorted by index; the first 40 are used for training and the rest are held out, with no random assignment. Held-out windows are formed like training windows (Appendix C).

Table 12: Per-interface drift of earlier tasks on LIBERO-Object, values from the end-of-task checkpoint analysis (training seed 0) on fixed held-out windows (mean over all pairs $k < l$). *modal*: change of the per-step modal tokens; *fusion* $\mid e$: change of the fused latent with the input held at task $k$'s tokens; *head KL* $\mid g$: reverse KL of the policy head with the input held at task $k$'s latent; the last three columns are full-path changes. $\ell_2$ values are Euclidean norms; KL is the reparameterized Monte Carlo estimate.

| Method | modal $\ell_2$ | fusion $\ell_2 \mid e$ | head KL $\mid g$ | latent $\ell_2$ | policy KL | action $\ell_2$ |
|---|---|---|---|---|---|---|
| Sequential | 0.403 | 0.183 | 0.055 | 0.587 | 0.257 | 0.220 |
| M2Distill-R | 0.290 | 0.132 | 0.040 | 0.422 | 0.185 | 0.158 |
| PathComp-fresh | 0.250 | 0.114 | 0.034 | 0.364 | 0.159 | 0.136 |
| PathComp | 0.194 | 0.088 | 0.062 | 0.282 | 0.123 | 0.106 |

**Shared evaluation windows.** For each suite, one fixed permutation selects held-out windows for every task. The permutation is seeded by the suite, task file name and evaluation-protocol version, independently of the training seed, method and task order. Every run evaluates the same first 192 selected windows, in sorted order, after each task. The selected identifiers and demonstration split are retained with the results so that comparisons use identical examples.

**Metrics.** $E_{l,k}$ is the error on task $k$ after training task $l$. It is the squared error between the decoded and the demonstrated action at the last step of a window, summed over the 7 action dimensions and averaged over the 192 windows. Final error, forgetting and immediate error are defined from $E_{l,k}$ in Section 4.1. These are offline metrics; closed-loop success is not measured for this policy.

**Step-0 regression test.** At the first task every memory, anchor buffer and teacher is empty, so every objective reduces to behavior cloning. A CPU test runs one update of every configuration used in the paired runs from the same seed, initialization and batch, and asserts that the parameters equal those of Sequential up to floating-point tolerance. A second test checks that the updates do differ once memory exists, so the first test cannot pass vacuously.

**Causal mask.** An earlier version of the temporal module used an attention mask that let each step attend to later steps. All reported paired runs follow the corrected causal-mask configuration. Each paired run checks the mask at startup and refuses to start if the check fails; earlier pilot runs predate this startup check. Runs made before the fix are excluded entirely.

**Recorded evaluation details.** The reproducibility materials include the configurations, software versions, task order, demonstration split, evaluation-window identifiers and per-seed error matrices. Each matrix cell is accompanied by its evaluation count; timing and loss components are recorded separately. The paired comparisons use completed runs with the prescribed architecture, causal mask, training schedule and shared evaluation windows.

**What is excluded, and why.** Runs made before the causal-mask fix are excluded. Single-seed pilots scored by the legacy unpaired evaluator, which drew its own random held-out windows for each run, support no claim and are not reported; they include every run of an earlier M2Distill variant (all weights 1.0, no replay memory). In one such pilot, task-1 errors differed across methods although every objective reduces to behavior cloning on the first task; this motivated the shared evaluation windows and the step-0 test above. Short test runs used to check the implementation are not reported. For OpenVLA-7B, a preempted training seed is excluded (Appendix G).

**Closed-loop success on LIBERO.** We report no closed-loop LIBERO success rate for the CLIP–LoRA–GMM policy. Success rates measured during earlier development are not reused: their rollout logs are not available to us, and the present evaluation measures offline error.

# G OpenVLA-7B Feasibility Diagnostic

This appendix gives the setup, the confounds and the statistics behind Table 13. The diagnostic asks only whether the cached latent and policy terms run on a 7B VLA; Appendix A.4 states what its anchor terms bound.

**Starting point and training.** We start from an OpenVLA-7B checkpoint (Kim et al., 2025) fine-tuned on LIBERO-Object and fine-tune it on three LIBERO-Goal tasks for 1200 steps. LoRA adapters of rank 16 (LoRA alpha 32, dropout 0.1) sit on the query, key, value, output, gate, up and down projections of the language model. The vision tower, projector, token embeddings and output head stay frozen. Each step uses 4 Goal examples, AdamW with learning rate $10^{-4}$ and gradient-norm clipping at 1. An example is one agent-view frame resized to $224 \times 224$, the instruction prompt and the 7 discretized action tokens. Training frames come from the first 40 demonstrations of each task, and frames without arm motion or gripper change are dropped.

**Cached anchors.** Before training, the starting model in evaluation mode caches 180 items, 45 from each of the first four Object tasks in alphabetical file order (alphabet soup, BBQ sauce, butter, chocolate pudding), drawn without replacement from the training frames. For each item it stores the action-token distributions and last-layer hidden states at the positions that predict the 7 action tokens, and the mean entropy of those distributions. It also stores the preprocessed image and the prompt with the demonstrated action tokens, because the items are replayed through the model; this instantiation is therefore not raw-free. The vision tower is frozen, so there is no modal term.

**Objective.** Each step adds 3 cached items, drawn round-robin over the four Object tasks and uniformly within a task. Each item contributes an un-gated rehearsal cross-entropy on its stored action tokens plus $\lambda_{\mathrm{un}}\big(\lambda_\pi \ell_\pi + \lambda_{\mathrm{lat}} \ell_{\mathrm{lat}}\big)$. Here $\ell_\pi$ is the forward KL from the cached to the current action-token distributions, averaged over the 7 positions, and $\ell_{\mathrm{lat}}$ is the mean squared error of the hidden states. We use $\lambda_\pi = 0.1$, $\lambda_{\mathrm{lat}} = 0.05$ and $\lambda_{\mathrm{un}} = \min(\exp(-\gamma \bar{\mathsf{H}}), 1)$ with $\gamma = 0.5$, where $\bar{\mathsf{H}}$ is the item's mean entropy. The uniform cached-anchor baseline uses the same items, rehearsal and policy term with $\lambda_{\mathrm{un}} = 1$ and $\lambda_{\mathrm{lat}} = 0$; it is not M2Distill. Sequential fine-tunes on the Goal examples alone. All anchor terms are computed with LoRA dropout active (Appendix A.4). The targets are cached by the starting model, which is also the previous model of this single stage, so acquisition-time and previous-model targets coincide and persistence is not tested.

**Task sets.** The training script selects Goal tasks and anchor tasks by position in the alphabetically sorted list of demonstration files, while the rollout script selects tasks by LIBERO's benchmark index, and the two orders differ. Retention rollouts cover Object benchmark tasks 0–2 (alphabet soup, cream cheese, salad dressing), so only alphabet soup contributes cached items. Goal rollouts cover benchmark tasks 0–2 (open the middle drawer, put the bowl on the stove, put the wine bottle on top of the cabinet), so only the first was trained; the other two trained tasks (open the top drawer and put the bowl inside, push the plate to the front of the stove) were never rolled out. Over the four seeds, the cached task gives 6, 9 and 5 successes in 65 trials for PathComp, Sequential and the uniform baseline, and the two other tasks give 26, 19 and 14 in 130. Training and rollout task lists must be aligned to interpret retention and acquisition on the same tasks.

**What the diagnostic cannot show.** Six differences from the LIBERO protocol limit it. (i) There is one stage, so acquisition-time and previous-model targets coincide and target persistence is not tested. (ii) Anchors replay the preprocessed pixels, the prompt and the demonstrated action tokens under an un-gated rehearsal cross-entropy, so the instantiation is replay plus distillation, not raw-free. (iii) The policy term is a forward KL on action tokens rather than the reverse KL on a GMM of Section 3.3, and every anchor term runs with LoRA dropout active. (iv) Only one of the three Object tasks rolled out for retention has cached anchors, and only one of the three Goal tasks rolled out for acquisition was trained (*Task sets* above). (v) The uniform cached-anchor arm differs from PathComp in both the confidence weight and the latent weight ($\lambda_{\mathrm{lat}} = 0$ versus $0.05$), so the pair does not isolate confidence weighting. (vi) The closed-loop arms come from at least two versions of the training script (*Script versions* below). On the one trained Goal task that was rolled out, no arm succeeds in more than 1 of 15 seed-0 rollouts, so the diagnostic does not show continual acquisition either.

Table 13: OpenVLA-7B feasibility diagnostic (cached-anchor instantiation, one Goal stage, so target persistence is not tested): closed-loop success on 3 LIBERO-Object tasks, of which only one has cached anchors, after LoRA fine-tuning on 3 LIBERO-Goal tasks; anchors replay raw pixels and text (*not* raw-free). *Uniform cached anchors*: unit weights, no latent term (not M2Distill). Seeds 0–3 (seed 0: 20 rollouts per task, seeds 1–3: 15; base: seed 0). Seed mean: mean±std of per-seed success rates. PathComp − row: mean over the 12 (seed, task) cells with a 95% seed-cluster bootstrap interval (the four training seeds resampled; coarse with four clusters). Seeds +/−/=: training seeds in which PathComp's success rate is higher, lower or equal. Descriptive only. Goal, trained: the one rolled-out Goal task that was trained, seed 0.

| Method | Obj. succ./trials | Seed mean | PathComp − row [95% seed-cluster CI] | Seeds +/−/= | Goal, trained (s0) |
|---|---|---|---|---|---|
| Base checkpoint | 37/60 (0.617) | – | – | – | 0/15 |
| Sequential | 28/195 (0.144) | 0.151 ±0.153 | +0.001 [−0.178, +0.182] | 2/2/0 | 1/15 |
| Uniform cached anchors | 19/195 (0.097) | 0.090 ±0.081 | +0.063 [+0.017, +0.104] | 3/0/1 | 1/15 |
| PathComp | 32/195 (0.164) | 0.153 ±0.100 | – | – | 0/15 |

Table 14: Instruction-paraphrase sanity check of the base OpenVLA-7B LIBERO-Object checkpoint (before any continual training): closed-loop success on 3 Object tasks with 20 rollouts per task, canonical LIBERO instructions vs. our LLM-generated paraphrases. One checkpoint, descriptive.

| Instructions | Successes/trials (rate) |
|---|---|
| Canonical | 37/60 (0.617) |
| LLM-paraphrased | 34/60 (0.567) |

**Closed-loop evaluation.** Each rollout starts from LIBERO's fixed initial state for its index and runs for at most 400 steps with greedy decoding in evaluation mode; it succeeds if the benchmark's task predicate holds. Training seed 0 has 20 rollouts per task and seeds 1–3 have 15, so each method has 195 Object trials in 12 (seed, task) cells. The base checkpoint was rolled out in seed 0 only, with 60 trials. Goal success was measured in seed 0, with 15 rollouts per task. A fifth training seed was preempted before its rollouts finished and is excluded.

**Statistics.** Rollout files store per-task success rates. We recover successes as the rate times the trial count taken from the launch scripts, and check that the product is an integer. A comparison averages the per-cell success-rate difference over the 12 common (seed, task) cells. The three cells of a seed share one trained model, so each arm has only four independent units. The reported 95% interval therefore resamples whole training seeds (percentile interval, 10,000 bootstrap resamples), and with four seeds it is coarse. Against Sequential, the mean difference is +0.001 [−0.178, +0.182]. The per-seed differences are +0.250, +0.022, −0.022 and −0.244 for seeds 0–3, so PathComp is higher in 2 seeds and lower in 2; the diagnostic does not distinguish PathComp from Sequential. Against the uniform cached-anchor arm, the mean difference is +0.063 [+0.017, +0.104], with per-seed differences +0.117, +0.067, +0.067 and +0.000: higher in 3 seeds, lower in 0 and equal in 1. Most of that gap lies on the two tasks without cached anchors (*Task sets* above), and confound (v) keeps us from attributing it to confidence weighting. We report no p-value: a sign test over cells or a pooled two-proportion test would treat the cells or rollouts of one trained model as independent. All of these statistics are descriptive.

**Script versions.** The closed-loop runs come from at least two versions of the training script, and within seed 3 the PathComp and uniform arms were produced by different versions. That both arms cache the same items is verified only for the current version, which we will release and whose selection seed does not depend on the method.

## H Instruction-Paraphrase Set: Datasheet

We will release GPT-4o paraphrases of LIBERO instructions to test sensitivity to instruction wording.

**Intended use.** The set is for evaluation only: a paraphrase replaces the canonical instruction while the scene and initial state stay fixed. It is neither training data nor a language-understanding benchmark.

**Composition.** The set contains 240 paraphrases: 12 for each of the ten Object and ten Goal tasks. A JSON file maps each suite and exact canonical demonstration instruction to strings distinct up to letter case. For "pick up the alphabet soup and place it in the basket," examples include "Grab the alphabet soup and place it in the basket." and "Move the alphabet soup to the basket." The set contains no personal data.

**Generation and validation.** Canonical instructions came from demonstration metadata. GPT-4o, at temperature 0.9, was asked for 12 paraphrases preserving objects and goals while varying unambiguous synonyms, word order, polite or imperative forms, verbosity and colloquial register. Responses had to parse as JSON arrays of 12 distinct, non-empty strings; failures were retried up to three times at slightly higher temperatures. All 20 tasks passed. Validation was automatic, without human judgments of semantic equivalence. Paraphrases can shift meaning: "move," for example, need not imply picking up. The English-only set covers two suites but was evaluated here on three Object tasks.

**Use in this paper.** Table 14 evaluates the base OpenVLA-7B checkpoint before continual training on three Object tasks, with 20 rollouts per task and paraphrase index $r \bmod 12$ for rollout $r$. Paraphrased instructions yield 34/60 successes versus 37/60 with canonical wording. These are separate rollout runs of one checkpoint, a descriptive comparison that supports no continual-learning claim. No continually trained model was evaluated with paraphrases.

**Distribution and license.** The file and datasheet will accompany the code after acceptance under CC BY 4.0, with attribution to LIBERO (Liu et al., 2023), whose source demonstrations use the same license.

# I Evaluation Scope and Remaining Questions

The primary evidence comes from offline action prediction with one CLIP–LoRA–GMM policy, one demonstration split and three training seeds on LIBERO-Object. Goal and Spatial use two seeds; the reversed order, joint reference and interface diagnostic use one. Anchor-window selection is fixed across seeds. These comparisons indicate consistency within the evaluated setting, while additional task orders, independently sampled memories and closed-loop evaluations would test broader generalization. The separate OpenVLA diagnostic has protocol differences and does not establish a closed-loop retention advantage.

The comparisons share the policy, training schedule, stored windows and evaluation inputs. They do not match tuning effort, replay batch size or compute. M2Distill-R achieves better likelihood despite PathComp's lower decoded-action MSE, so the relative benefit depends on the prediction metric. The component ablations test the feature and policy constraints, removal of the modal anchor and the confidence-weighting rule. They do not isolate removal of the latent anchor while retaining both other terms. Uniform weights also change average regularization strength; that comparison evaluates the complete weighting rule.

The drift analysis applies to predictive distributions on stored anchors, rather than directly bounding held-out action error or closed-loop success. The implemented policy estimator has no direct expected gradient to mixture logits, although behavior cloning and shared features can update them. Full PathComp uses replay-scale storage; removing raw windows leaves encoder drift unconstrained. The memory quota assumes a known task count, and eviction is inactive in the evaluated ten-task streams.