SCALING CONTINUAL LEARNING TO 125 TASKS WITH EXPERIENCE REPLAY

Anonymous authors
Paper under double-blind review

ABSTRACT

The ability to continually acquire new skills while retaining existing capabilities is important for the long-term deployment of vision-language-action (VLA) models. However, scaling VLA continual learning beyond a hundred tasks remains underexplored. Prior work mainly studies task-wise updates on small task sets. Scaling to task pools of this size requires incorporating new capabilities in batches, motivating phase-wise continual learning, where each update jointly learns a batch of new tasks. We systematically study which continual learning recipes scale to this setting. Experiments on LIBERO and RoboCasa365 show that task-wise findings do not directly transfer to phase-wise updates. For example, adding low-rank adaptation (LoRA) to experience replay can benefit some task-wise settings through better old-task retention, but this advantage does not reliably persist across phases. Higher ranks and tuned adapter learning rates improve acquisition, yet LoRA still trails full-model experience replay and can forget more of the capabilities learned in intermediate phases. Further analysis shows that task composition and update granularity shape learning dynamics. We find that simple full-model experience replay (ER) effectively balances new-task acquisition and old-task retention in phase-wise VLA continual learning, making it the most practical choice among the recipes we evaluate. Across four phases and 125 tasks in RoboCasa365, full-model ER continually acquires new capabilities while preserving old ones, using only 2% of each old task's data for replay and requiring neither additional teacher training nor online data collection. Our results establish full-model ER as a practical, scalable recipe for phase-wise VLA continual learning.

1 INTRODUCTION

Vision-Language-Action (VLA) models are becoming an important foundation for general-purpose robot policies, but their training and iteration costs grow rapidly with model and data scale (Intelligence et al., 2025). As new tasks, environments, and behavioral requirements continually emerge, retraining a VLA model from scratch for every update is impractical; instead, continual learning (CL) is needed so that a model can absorb new capabilities while preserving existing ones. Existing studies on VLA continual learning have primarily focused on task-wise protocols, where each update introduces a single new task (Liu et al., 2026; Zhu et al., 2026; He et al., 2026). At the scale of 125 tasks, organizing continual learning around isolated single-task updates is impractical; models need to incorporate batches of new capabilities. We refer to updates that learn such batches as *phase-wise continual learning* (phase-wise CL). We study how VLA continual learning can scale to 125 tasks and which CL recipes remain effective at this scale. A first question is therefore: **can conclusions from task-wise VLA continual learning be directly transferred to phase-wise CL?** Our experiments show that the answer is **no**. On LIBERO (Liu et al., 2023) and RoboCasa365 (Nasiriany et al., 2026), the tested task-wise and phase-wise streams yield different acquisition and forgetting curves. Experiments with small task mixtures show that some new tasks support old-task retention within particular mixtures. Controlled comparisons with fixed task sets and training budgets show that larger task groups improve both acquisition and retention. These findings reinforce the importance of phase-wise updates for scaling VLA continual learning. We further compare common continual learning strategies and find that their relative benefits also depend on the update protocol. Adding LoRA (Hu et al., 2021) to Experience Replay (ER) (Chaudhry et al., 2019) can help in some task-wise settings by substantially improving old-task retention, and several ER-based VLA CL methods adopt this combination as their default configuration (Chen et al., 2026; Liu et al., 2026; He et al., 2026; Syed et al., 2026). In the first phase-wise update,

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.

full-model ER already forgets little, narrowing LoRA's retention advantage, while low-rank LoRA remains acquisition-limited across tested adapter learning rates. Higher ranks and tuned adapter learning rates narrow this gap, but high-rank LoRA with a higher adapter learning rate forgets more of the capabilities learned in intermediate phases. Preserving initial capabilities therefore does not guarantee durable retention of newly acquired ones.

Full-model ER provides a strong and scalable recipe for VLA continual learning, retaining old capabilities while acquiring new ones across 125 tasks with only 2% of old-task data. ER can approach the performance of separately trained phase-specific teachers without training an additional teacher or collecting online rollouts. Our comparisons with model merging, reinforcement learning, and policy distillation show that these additional components are not necessary for strong performance. Our main contributions are summarized as follows:

- We systematically study VLA continual learning at a scale of 125 tasks under phase-wise updates, a practical but under-explored setting where each update introduces a batch of new tasks rather than a single task.
- We provide counterexamples showing that task-wise findings do not always transfer to phase-wise VLA CL, analyze how task composition and update granularity shape acquisition and retention dynamics, and identify ER as a strong and simple recipe after evaluating common continual learning strategies.
- We demonstrate effective continual learning across 125 tasks with full-model ER, without additional teacher training or online data collection.

2 Related Work

Continual learning for robot policies. Classical continual learning methods mitigate forgetting through parameter regularization (Kirkpatrick et al., 2017), output distillation (Li & Hoiem, 2017), and experience replay (Chaudhry et al., 2019). Continual robot learning also explores task-specific adapters (Liu et al., 2024) and reusable skills discovered from demonstrations (Wan et al., 2024). Continual learning for VLA policies has recently received growing attention. Existing studies mainly focus on task-wise CL, where each update adds one new task. Liu et al. (2026) shows that ER with only 2% old-task data can already work well, including a π_0 configuration that uses LoRA for its language and action modules. Other studies combine ER with LoRA or partial parameter updates (He et al., 2026; Zhao et al., 2026; Chen et al., 2026). In language models, LoRA preserves initial capabilities well (Biderman et al., 2024), yet can forget more under repeated adaptation (Shuttleworth et al., 2025). VLA CL methods also use specialized memory or replay selection, expert, routing, or knowledge-space designs (Du et al., 2026; Wu et al., 2026; Syed et al., 2026; Römer et al., 2026). In conventional CL, replay selection has been guided by gradient diversity (Aljundi et al., 2019b) and predicted interference from incoming updates (Aljundi et al., 2019a). However, it remains unclear whether the common task-wise findings still hold in the phase-wise setting, where each update adds a batch of new tasks. We do not claim phase-wise CL as a new concept; conventional class-incremental studies similarly find that update granularity affects learning dynamics, but this question remains under-explored in VLA CL (Rebuffi et al., 2017; Douillard et al., 2020; 2021).

Model merging and policy distillation. Model merging provides a simple way to combine knowledge from different fine-tuned models. In VLA policies, Yadav et al. (2026) shows that parameter interpolation can improve robust fine-tuning and continual skill acquisition. In language and vision models, recent SVD-based merging methods further show that task updates contain useful low-rank or spectral structure that can reduce merging interference (Stoica et al., 2024; Gargiulo et al., 2025). On-policy distillation (OPD) uses teacher supervision on states sampled from the student policy, and has recently been extended to flow matching models and VLA post-training (Lu & Lab, 2025; Fang et al., 2026; Zhong et al., 2026). For robot policies, refreshing on-policy data adds environment-interaction cost beyond offline training. Online RL methods incur this cost as well (Hu et al., 2026). We compare policy distillation with other updates built on ER, including a fixed-buffer variant of Flow-OPD that reuses a small rollout batch for multiple updates.

3 Study Setup and Compared Methods

We describe the continual learning setting and ER, then introduce model merging and policy distillation to test whether more complex updates are necessary.

3.1 Problem Formulation

We consider continual learning for a VLA policy π_θ, which maps an observation o and a language instruction l to an action chunk x_0. At each update, the model receives new demonstrations and must learn new tasks while preserving previous ones.

Task-wise and phase-wise continual learning differ in the granularity of each update. In task-wise CL, update i introduces one new task τ_i with data $\mathcal{D}_{\tau_i}$. In phase-wise CL, phase k introduces a task batch $\mathcal{P}_k$ with data $\mathcal{D}_k = \cup_{\tau \in \mathcal{P}_k} \mathcal{D}_\tau$. These two protocols can be written as

$$\text{task-wise:} \quad \pi_{\theta_i} = \mathcal{A}\left(\pi_{\theta_{i-1}}, \mathcal{D}_{\tau_i}\right), \text{phase-wise:} \quad \pi_{\theta_k} = \mathcal{A}\left(\pi_{\theta_{k-1}}, \mathcal{D}_k\right), \tag{1}$$

where $\mathcal{A}$ denotes a continual learning algorithm. After each update, the model is evaluated on all seen tasks: $\{\tau_1, \ldots, \tau_i\}$ for task-wise CL and $\cup_{j=1}^{k} \mathcal{P}_j$ for phase-wise CL.

Both settings aim to acquire new tasks while retaining previously learned capabilities as the task pool grows. Our study focuses on phase-wise CL and reports success on each seen phase separately after every update, assessing both new-phase acquisition and old-phase retention.

3.2 Experience Replay SFT

ER serves as the base for our comparisons. When training on phase k, we mix the new-task data $\mathcal{D}_k$ with a small replay buffer from previous tasks. Following prior work (Liu et al., 2026), we keep only 2% of the data for each old task:

$$\mathcal{B}_{<k} = \bigcup_{\tau \in \mathcal{T}_{<k}} \text{Sample}\left(\mathcal{D}_\tau, \rho\right), \quad \rho = 2\%. \tag{2}$$

Here $\mathcal{T}_{<k} = \bigcup_{j<k} \mathcal{P}_j$ is the set of previously introduced tasks. During training, old-task replay samples take 31.25% of each batch, and the remaining samples come from the current phase. We use the same flow matching (Lipman et al., 2023) objective as standard VLA supervised fine-tuning. Let x_0 be the clean action chunk, $\epsilon \sim \mathcal{N}(0, I)$ be Gaussian noise, and $t = 0$ denote clean data while $t = 1$ denotes noise:

$$x_t = (1-t)x_0 + t\epsilon, \quad \mathcal{L}_{\text{FM}} = \mathbb{E}_{(o,l,x_0),\epsilon,t} \left\| v_\theta(x_t, t, o, l) - (\epsilon - x_0) \right\|_2^2. \tag{3}$$

During training, $t = 0.001 + 0.999u$ with $u \sim \text{Beta}(1.5, 1)$. Here o is the observation and l is the language instruction; sampling runs from $t = 1$ to $t = 0$. Our experiments show that ER alone is already a strong recipe for phase-wise VLA CL. With only 2% old-task data, it can learn the new phase well while causing little forgetting on previous phases. We further test whether keeping a small learning rate and training for a short period after cosine decay improves ER, using the same objective without an additional teacher or online rollouts.

3.3 Model Merge

Dense merge combines two models by averaging their parameters. For our teacher-assisted comparisons, we train a per-phase teacher π_{teacher} from the pretrained $\pi_{0.5}$ checkpoint using only the current phase data. Let π_{student} be the continual student before phase k. We average the student and teacher parameters in proportion to the numbers of phases they represent, then either evaluate the merged model directly or use it to initialize ER SFT.

SVD merge retains a low-rank part of the parameter update. Both merge methods use the same phase-based weighting:

$$\alpha = \frac{1}{k}, \quad \theta_{\text{mixed}} = (1-\alpha)\theta_{\text{student}} + \alpha\theta_{\text{teacher}}, \tag{4}$$

where the student has learned the preceding $k-1$ phases and the teacher is trained on phase k. For the attention and feed-forward projection matrices in the vision-language backbone and action expert, we retain the leading singular components of the update:

$$\Delta W = W_{\text{mixed}} - W_{\text{student}}, \quad \Delta W_r = \text{SVD}_r(\Delta W), \quad W_{\text{svd}} = W_{\text{student}} + \Delta W_r. \tag{5}$$

We use rank 128 in the vision-language backbone and rank 256 in the action expert, leave parameters outside these projections unchanged, and use the merged model to initialize full-model ER SFT.

RETAIN (Yadav et al., 2026) applies parameter interpolation to the models before and after fine-tuning, rather than to a student and a separate phase teacher. In our comparison, we interpolate the pre-update and ER-updated parameters with 80% weight on ER and 20% on the pre-update checkpoint, then evaluate without further training. Appendix B.6 reports the full mixing-weight sweep. RETAIN therefore retains the full training cost of ER and adds checkpoint merging, offering no simplification of the training pipeline.

3.4 POLICY DISTILLATION

Policy distillation uses teacher supervision to train the student. We evaluate direct Flow-OPD from the pre-update model as well as distillation after ER SFT. On-policy distillation (OPD) uses states sampled by the current student. Conditioning on the observation o and language instruction l is omitted for brevity. Following the flow-policy formulation of Flow-OPD (Fang et al., 2026), each denoising step is represented as a Gaussian transition policy:

$$p_\theta(x_{t+\Delta t} \mid x_t, t) = \mathcal{N}\left(b(x_t, t) + a(t)v_\theta(x_t, t), \sigma(t)^2 I\right), \tag{6}$$

where $\Delta t < 0$ for denoising, $b(x_t, t)$ is the part of the transition mean that does not depend on the learned velocity field, $a(t)$ maps the velocity field to the transition mean, and $\sigma(t)$ is the transition noise standard deviation.

Under this form, the student and teacher share the same covariance and differ only in their velocity fields. Let d_π denote the joint sampling distribution of noisy actions and denoising times obtained from rollouts of policy π under the training sampling procedure. The OPD loss is therefore

$$\mathcal{L}_{\text{OPD}} = \mathbb{E}_{(x_t,t)\sim d_{\text{student}}}\left[\frac{a(t)^2}{2\sigma(t)^2}\left\|v_{\text{student}}(x_t, t) - \text{sg}\left(v_{\text{teacher}}(x_t, t)\right)\right\|_2^2\right], \tag{7}$$

where sg denotes stop-gradient.

We also consider iterative policy distillation (IPD): the student first collects a rollout buffer, which is then reused for 1,000 optimization steps. These updates use the Flow-OPD loss in Equation 7. During IPD, we still mix in replay data from old tasks.

4 PHASE-WISE CONTINUAL LEARNING

We first study whether conclusions from task-wise VLA continual learning transfer to the phase-wise setting. We compare their acquisition and forgetting behavior, evaluate common continual learning methods under phase-wise updates, and examine how task composition and update granularity shape learning dynamics.

Setup. We use $\pi_{0.5}$ as the VLA backbone and study LIBERO and RoboCasa365. We index the ten LIBERO-10 tasks as B0–B9 and first train B0–B4. Starting from the same B0–B4 checkpoint, the task-wise arm trains only B5, whereas the phase-wise arm jointly trains B5–B9. Both arms use dense full-parameter SFT for 3k updates, with a fresh optimizer, the same batch size, and the same learning-rate schedule. Each LIBERO-10 success rate is measured over 50 benchmark-provided initial states per task. On RoboCasa365, we first train the 65 Phase1 tasks. The task-wise arm then trains only `TurnOffSimmeredSauceHeat`, while the phase-wise arm trains all 20 Phase2 tasks. These arms also start from the same Phase1 checkpoint and use a fresh optimizer, the same batch size, and the same learning-rate schedule. RoboCasa main experiments use 128 evaluation episodes per task by default, with task success rates averaged equally within each reported task group.

We compare sequential SFT, ER, and their LoRA variants, together with dense merge, RETAIN, direct Flow-OPD, and Flow-GRPO (Liu et al., 2025). For RoboCasa ER with LoRA, $1\times$ and $5\times$ scale only adapter learning rates; other trainable modules retain the reference schedule. Dense merge and RETAIN are evaluated directly after the parameter interpolation described in Section 3, without further training. Direct Flow-OPD uses the Flow-OPD loss, and Flow-GRPO is implemented with RLinf (Yu et al., 2025). Both update the Phase1 base directly and include replay, with 352 new and 160 old samples per batch.

4.1 TASK-WISE CONCLUSIONS DO NOT DIRECTLY TRANSFER TO PHASE-WISE CL

Figure 1 compares task-wise and phase-wise sequential SFT on both benchmarks, showing different acquisition and retention dynamics. Under the tested task streams, task-wise SFT rapidly loses old-task performance, whereas phase-wise SFT retains non-trivial old-task success. On LIBERO-10, acquiring the full five-task set is more demanding than mastering B5 alone. Nevertheless, the shared new task B5 reaches 100% under both protocols, so the retention gap is not caused by a failure to acquire B5. Comparisons over fixed task sets and training budgets further show that update granularity affects acquisition and retention (Section 4.3).

The benefit of adding LoRA to ER also depends on the protocol. Table 1 compares sequential SFT, ER, and ER with LoRA at the reference learning rate. In the task-wise update, adding LoRA to ER trades some new-task acquisition for substantially better retention over the 65 old tasks, making it favorable in some single-task CL settings. In this phase-wise update, its additional retention benefit is small, while the loss in new-task acquisition is much larger. These comparisons already use LoRA

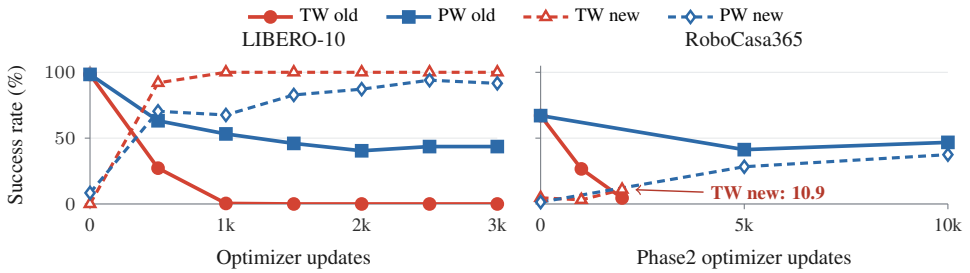


Figure 1: Task-wise (TW) and phase-wise (PW) acquisition and forgetting under sequential SFT. Old-task success is averaged over previous tasks; new-task success is averaged over tasks introduced by each protocol. The horizontal axes report continual-update steps for LIBERO-10 and Phase2 steps for RoboCasa365.

ranks of 128 for the language model and 256 for the action expert, well above the ranks of 16–32 used in prior VLA CL studies (He et al., 2026; Chen et al., 2026). At these ranks, increasing the adapter learning rate fivefold substantially narrows the acquisition gap, but new-task success remains below full-model ER (Figure 2). Rank and learning-rate ablations are reported in Appendix B.7. Full-model ER therefore gives a better balance, showing that the task-wise benefit of ER with LoRA does not transfer reliably to phase-wise CL.

We additionally report backward transfer (BWT) and forward transfer (FWT), two widely used CL metrics defined in Appendix A.5. FWT is measured at the shared Phase1 starting point, before the compared update methods are applied.

Table 1: Task-wise and phase-wise RoboCasa365 success rates (%) and transfer metrics (pp). Phase1 base is evaluated before the update. Old tasks are the Phase1 tasks; new tasks are `TurnOffSimmeredSauceHeat` in the task-wise update and the Phase2 tasks in the phase-wise update.

	Task-wise (1 task)				Phase-wise (20 tasks)			
Metric	**Phase1 base**	**Sequential SFT**	**ER**	**ER + LoRA**	**Phase1 base**	**Sequential SFT**	**ER**	**ER + LoRA**
Old-task success rate (%)	67.1	4.5	56.7	**66.5**	67.1	43.6	67.3	**68.7**
New-task success rate (%)	4.7	10.9	**14.8**	10.9	1.4	31.2	**35.4**	17.1
BWT (pp)	–	−62.6	−10.4	−0.6	–	−23.5	+0.2	+1.6
FWT (pp)	+3.1	–	–	–	+1.3	–	–	–

4.2 Common CL Recipes in Phase-Wise CL

We next compare common continual learning methods on the RoboCasa365 Phase1-to-Phase2 setting. Figure 2 reports Phase1 retention and Phase2 acquisition for each method.

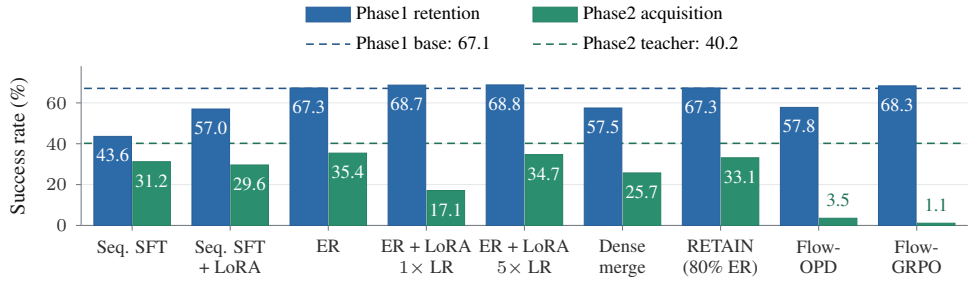


Figure 2: Common continual learning recipes on RoboCasa365 Phase1-to-Phase2. Bars report Phase1 retention and Phase2 acquisition; dashed lines show the Phase1 base and Phase2 teacher. LoRA multipliers apply only to adapter learning rates.

ER improves both old- and new-task success over sequential SFT, showing that replay can benefit new-task learning as well as retention. At the reference learning rate, LoRA shifts both sequential SFT and ER toward old-task retention, but reduces new-task acquisition, especially when combined with ER. At 5× adapter LR, LoRA approaches full-model ER's acquisition with slightly better retention in the first update, but this retention advantage does not persist across phases (Section 5.2). Dense merge also gives a weaker old/new balance, while RETAIN with 80% ER weight achieves similar old-task success but lower Phase2 acquisition than ER. The full coefficient sweep shows that lower ER weights incur a larger acquisition cost, while higher weights provide little additional retention benefit (Appendix B.6). Direct Flow-OPD and Flow-GRPO also show limited new-phase acquisition.

Takeaway 1: Task-wise conclusions are not guaranteed to transfer to phase-wise CL. Among the tested common recipes, ER provides the strongest balance between old-task retention and new-task acquisition.

4.3 What Shapes the Task-Wise-to-Phase-Wise Gap?

Task composition. We first examine whether task count alone explains the retention gap. Figure 3 compares three multi-task updates—B5+B6+B7+B8, B5+B6+B8+B9, and the full B5–B9 set—together with five task-wise runs that train B5 through B9 separately. All settings acquire their new tasks. Old-task performance collapses under B5+B6+B8+B9, whereas B5+B6+B7+B8 and the full update retain similar performance; among the task-wise runs, only B7 maintains non-trivial retention. Thus, B7 accounts for most of the retention gap in this stream.

B0 asks the robot to place alphabet soup and tomato sauce in a basket, while B1 places cream cheese and butter in a basket. B7 combines the alphabet soup from B0 and the cream cheese from B1 under the same two-object basket goal. Figure 4(b) shows that B0 and B1 are retained much better than B2–B4 after both B7-only and full B5–B9 training. We call a new task that supports old-task retention during an update a *support task*; B7 is such a task in this experiment.

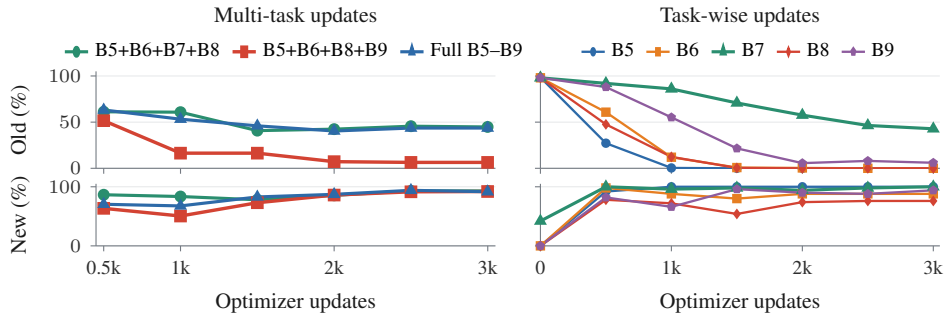


Figure 3: LIBERO-10 task composition. Top: old-task success rate; bottom: new-task success rate. The left panels compare joint training on B5+B6+B7+B8, B5+B6+B8+B9, and the full B5–B9 set. The right panels train B5 through B9 separately in task-wise updates.

We next test whether a similar support effect appears in RoboCasa365 using 2k SFT updates. We abbreviate `KettleBoiling`, `ClearCuttingBoard`, `TurnOffSimmeredSauceHeat`, and `PrepareCoffee` as Kettle, Clear, TurnOff, and Prepare. Phase1 retention is evaluated on 16 episodes for each of the 65 old tasks. Figure 4(a) reports the tested mixtures.

(a) RoboCasa365 task mixtures

Training tasks	P1 success (%)
Single-task updates	
Clear	25.38
Kettle	25.00
Two-task updates	
Prepare + Clear	**31.15**
TurnOff + Clear	30.87
TurnOff + Kettle	23.37
TurnOff + Prepare	15.10
Three- and four-task updates	
TurnOff + Kettle + Prepare	23.75
TurnOff + Prepare + Clear	9.62
TurnOff + Kettle + Prepare + Clear	23.17

(b) LIBERO-10 task success

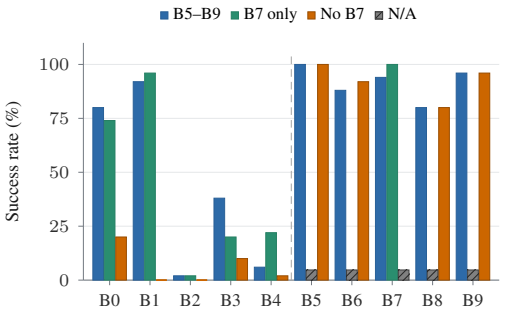


Figure 4: Support-task results. (a) RoboCasa365 Phase1 retention after 2k-step SFT without replay for selected Phase2 task mixtures. (b) LIBERO-10 per-task success for joint B5–B9, B7-only, and B5+B6+B8+B9 (No B7) training. The dashed line separates old and new tasks; gray stubs denote missing evaluations (N/A), with no quantitative height.

Kettle and Clear both retain substantial old-task performance in single-task and two-task training, whereas the TurnOff–Prepare pair is weaker. However, adding Clear to this pair decreases retention, while adding Kettle improves it. This difference is not explained by sampling frequency: Clear receives 33.16% of the samples in its three-task mixture, more than the 29.48% received by Kettle. Thus, the support effect depends on the training mixture and cannot be inferred from single-task performance alone. Full acquisition–retention results appear in Appendix B.9.

Kettle's single-task retention contrasts with the severe forgetting under TurnOff-only SFT (Table 1). Using this pair, we ask whether low-cost signals predict which old tasks benefit more from learning

Kettle than TurnOff. We test visual similarity between demonstrations, full-model gradient similarity, VLA vision-feature similarity, and the change in old-task loss after a 64-step update.

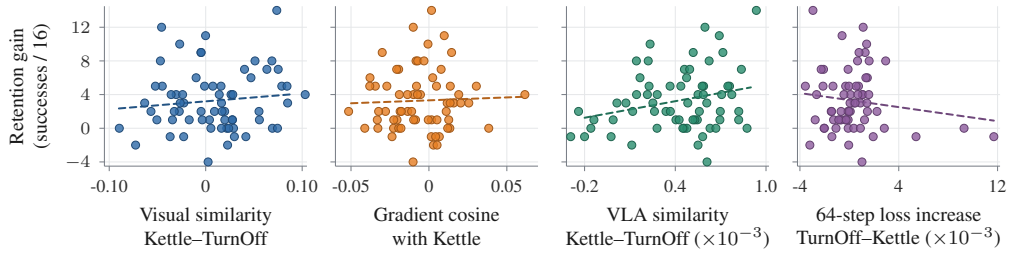


Figure 5: Low-cost proxies for the Kettle–TurnOff retention gap. Each point is one Phase1 task, and the vertical axis is its retained-success difference between training Kettle and TurnOff. From left to right, the horizontal axes show visual-similarity difference, gradient cosine with Kettle, VLA vision-feature similarity difference, and the difference in old-task flow-matching loss increase after 64 updates. Dashed lines are linear fits.

Using Pearson's r for the linear fits, Figure 5 shows that VLA features have the clearest, although still weak, positive trend ($r = 0.266$), followed by visual similarity ($r = 0.113$). Gradient similarity is nearly flat ($r = 0.040$), while the loss probe has a weak negative trend ($r = -0.140$). Overall, these semantic analyses do not yield a high-confidence correlation with old-task retention. Further analyses across all 20 Phase2 tasks and two LIBERO-10 screens likewise yield no consistent signal for identifying support-task candidates (Appendix B.3).

To see whether substantial retention is restricted to a few tasks, we train each of the 20 Phase2 tasks separately. Phase1 success averages 35.7%; 16 of the 20 models retain at least 30%, and the lowest achieves 11.9%. Although single-task retention does not establish support within a mixture, its prevalence suggests that updates involving more new tasks may be more likely to include tasks that preserve old capabilities.

Phase1-to-Phase2 FWT (Appendix A.5) provides a complementary view of task relationships. On LIBERO-10, the mean gain of 8.4 percentage points comes entirely from B7, consistent with its retention-support role. This correspondence does not generalize to RoboCasa365: Kettle and Clear have zero measured FWT, whereas TurnOff has $+3.125$ points despite its poor retention. Zero-shot FWT alone therefore does not reliably identify tasks that support retention in these experiments.

Update granularity. We compare SFT without replay, using batch size 128 and $50g$ steps per group of g tasks. RoboCasa learns the same 20 Phase2 tasks in groups of 1, 5, 10, or 20 (1,000 steps total); LIBERO learns all ten LIBERO-10 tasks from the same LIBERO-90 base in groups of 1, 2, 5, or 10 (500 steps total). LIBERO also matches the exact per-task training samples and preserves AdamW state and a constant learning rate across group boundaries.

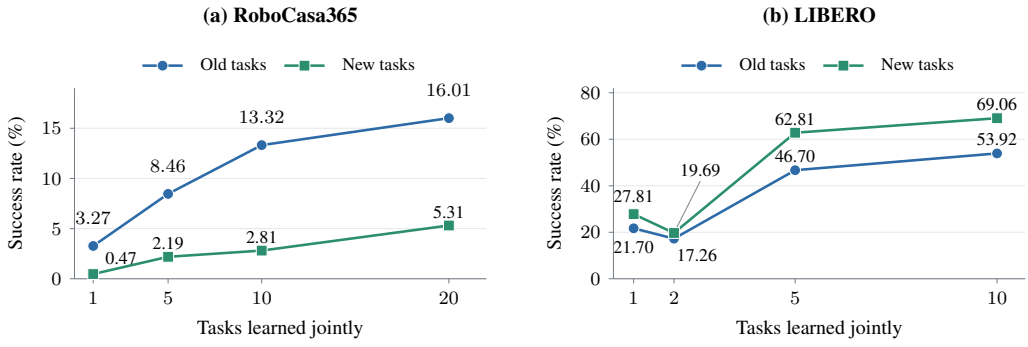


Figure 6: Success after all new tasks, without replay. (a) RoboCasa365: 20 tasks, 1,000 steps. (b) LIBERO: ten tasks, 500 steps. Task sets and budgets are fixed within each benchmark.

Across both benchmarks, larger task groups improve acquisition and retention under matched budgets, showing that update granularity changes learning dynamics. Phase-wise CL is driven by the need to incorporate new capabilities in batches when scaling to 125 tasks.

Takeaway 2: Task composition and update granularity shape acquisition and retention dynamics in VLA continual learning.

5 Scaling VLA Continual Learning to 125 Tasks

Building on the common-recipe comparison, we next ask whether ER alone can approach teacher performance and support continual learning across 125 tasks.

5.1 Approaching Teacher Performance with ER Alone

Section 4.2 identifies full-model ER as a strong starting point for phase-wise continual learning. Before scaling to the complete four-phase stream, we use the Phase1-to-Phase2 transition to examine how best to build on ER, since our compute budget does not permit a full four-phase evaluation of every alternative. We ask whether further ER training can approach teacher performance while retaining old skills, and whether model merging or rollout-based updates provide additional benefits. We first compare extending the original ER schedule with continuing ER at a low learning rate. ER15k and ER30k denote full-model ER trained for 15,000 and 30,000 optimizer steps, respectively. Under *No teacher*, ER+3k and ER+5k continue ER15k for 3,000 and 5,000 additional steps, respectively.

We then test whether additional teachers and online rollouts can improve on ER. Dense+ER and SVD+ER initialize 15,000 ER steps with dense and SVD model merge, respectively. All five methods under *Updates from SVD+ER* start independently from the same SVD+ER checkpoint. Here, ER+3k and ER+5k continue ER training; Flow-GRPO applies reinforcement learning, while Flow-OPD and IPD apply policy distillation to this checkpoint.

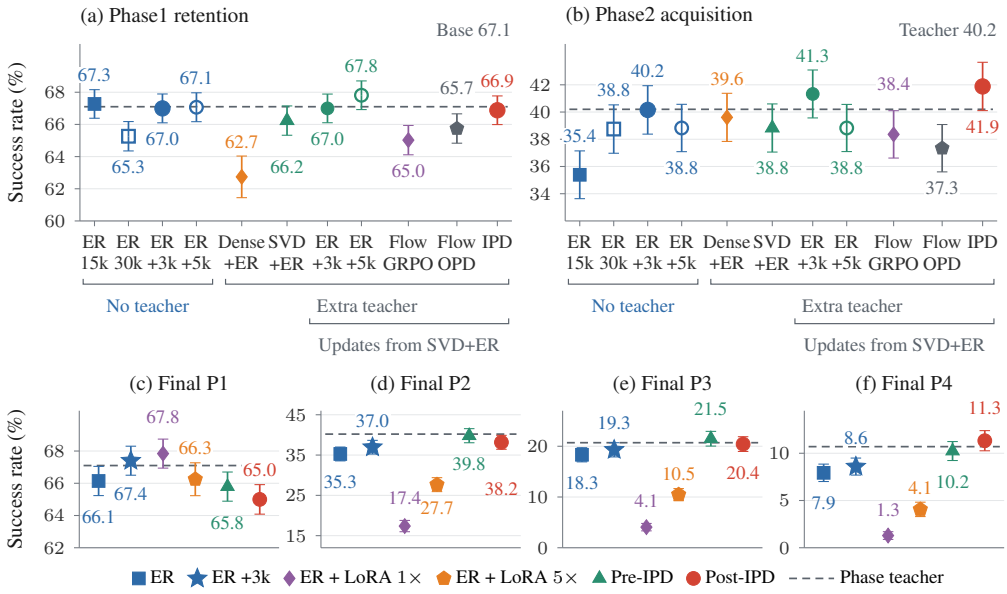


Figure 7: ER and teacher-assisted alternatives. (a,b) Phase1-to-Phase2 results; +3k/+5k denote low-learning-rate ER continuation. Merge-based methods require a teacher; Flow-GRPO, Flow-OPD, and IPD also collect online rollouts. (c–f) Phase4 results (Table 2; LoRA adapter LR 1×: Appendix B.8); both IPD checkpoints include IPD in Phases 2 and 3. Dashed lines mark the base or teacher; error bars show 95% evaluation confidence intervals (Appendix A.4.1).

ER+3k brings old- and new-task success close to the corresponding base and teacher (Figure 7(a,b)). Both scores exceed those obtained by directly extending ER training to 30,000 steps. Extending continuation to 5,000 steps does not further improve new-task success in either the plain ER or SVD-initialized ER run.

Among the updates from SVD+ER, IPD has the highest measured new-phase success, outperforming Flow-GRPO and Flow-OPD under our settings. IPD reuses a collected buffer for offline updates, whereas Flow-GRPO and Flow-OPD alternate parameter updates with fresh on-policy rollouts.

Full-model ER with low-learning-rate continuation is therefore our preferred recipe for scaling, approaching teacher performance without additional teachers or online rollouts.

5.2 Scaling Across 125 Tasks

Setup. We use the complete RoboCasa365 lifelong learning benchmark with 125 tasks over four phases. Phase1 contains 65 atomic tasks; Phases 2–4 each add 20 tasks with 2–3, 4–5, and 6+ stages, respectively (Nasiriany et al., 2026). This progression makes later phases more challenging, as reflected in the decreasing success of independently trained phase teachers (Table 2). After every update, we evaluate and report every seen phase separately. The ER baseline uses a batch size of 512 and 15k optimizer steps per continual update. We also apply ER+3k continuation at each continual update. LoRA starts from full-SFT Phase1 weights and uses $1\times$ or $5\times$ adapter LR, merging and reinitializing adapters between phases with ER.

Table 2: Per-phase success (%). ER+3k adds 3k continuation steps after each 15k ER update. Pre-/Post-IPD precede/follow the current phase's 1k IPD update in the same stream, including earlier IPD updates. BWT averages over old tasks before display rounding. Second lines give 95% confidence-interval half-widths (pp; Appendix A.4.1). LoRA uses $5\times$ adapter LR ($1\times$: Appendix B.8).

			After Phase2					After Phase3					After Phase4				
Eval.	**Teacher**	**Base**	**ER**	**ER +3k**	**ER + LoRA**	**Pre-IPD**	**Post-IPD**	**ER**	**ER +3k**	**ER + LoRA**	**Pre-IPD**	**Post-IPD**	**ER**	**ER +3k**	**ER + LoRA**	**Pre-IPD**	**Post-IPD**
P1	67.1 ±0.9	67.1 ±0.9	67.3 ±0.9	67.0 ±0.9	68.8 ±1.0	66.2 ±0.9	66.9 ±0.9	68.0 ±0.9	67.0 ±0.9	66.8 ±1.0	67.3 ±0.9	68.2 ±0.9	66.1 ±0.9	67.4 ±0.9	66.3 ±1.0	65.8 ±0.9	65.0 ±0.9
P2	40.2 ±1.8	–	35.4 ±1.8	40.2 ±1.8	34.7 ±1.8	38.8 ±1.8	41.9 ±1.8	35.8 ±1.7	37.4 ±1.8	31.1 ±1.8	37.1 ±1.7	40.0 ±1.7	35.3 ±1.7	37.0 ±1.7	27.7 ±1.7	39.8 ±1.7	38.2 ±1.7
P3	20.7 ±1.5	–	–	–	–	–	–	18.2 ±1.4	19.6 ±1.5	14.7 ±1.4	20.2 ±1.5	22.6 ±1.5	18.3 ±1.4	19.3 ±1.4	10.5 ±1.2	21.5 ±1.5	20.4 ±1.5
P4	10.7 ±1.0	–	–	–	–	–	–	–	–	–	–	–	7.9 ±0.9	8.6 ±0.9	4.1 ±0.8	10.2 ±1.0	11.3 ±1.1
BWT (pp)			+0.2	−0.1	+1.7	–	−0.2	+0.8	−0.7	−1.1	–	+0.4	−0.6	−0.5	−2.7	–	−2.4

The pre-/post-IPD comparison measures the current phase's IPD update along the same teacher-assisted stream, rather than comparing against an IPD-free training run (Table 2). ER+3k improves on ER15k across all four phases at the end of training, but still trails both Pre-IPD and Post-IPD on the later phases. The task-averaged BWT results in Table 2 show little net forgetting for ER and ER+3k. At $5\times$ adapter LR, LoRA approaches full-model ER's initial acquisition but remains weaker in later phases. Phase2 and Phase3 success subsequently drops by 7.0 and 4.2 pp, while full-model ER remains nearly unchanged. Preserving initial full-SFT capabilities therefore does not ensure retention of intermediate-phase capabilities. Successive low-rank updates may interfere (Appendix B.8). More elaborate adapter management might mitigate this interference, but full-model ER supports acquisition and retention without this complexity or LoRA-specific tuning. High-rank LoRA also approaches full-model ER's training compute (Appendix B.7). IPD further improves new-phase success at each transition, yet its Phase4 update reduces success on all three earlier phases (Figure 7c–f). Its acquisition gains therefore do not always yield a better retention–acquisition balance.

The full-stream results support the practical choice made from the Phase1-to-Phase2 comparison. Full-model ER continues to acquire new tasks with little net forgetting, and low-learning-rate continuation further improves its performance. Teacher-assisted updates achieve higher success on some phases, but require additional teacher training and online rollouts. Considering both performance and these requirements, ER remains our preferred practical recipe.

Takeaway 3: Full-model ER scales VLA continual learning to 125 tasks without additional teachers or online rollouts.

6 Conclusion

We study scalable VLA continual learning beyond a hundred tasks. Findings from small-scale task-wise settings do not directly transfer to phase-wise updates: task composition and update granularity shape learning and forgetting dynamics. Among the recipes we evaluate, simple full-model ER effectively balances new-task acquisition and old-task retention and scales to 125 tasks. Higher ranks and tuned adapter learning rates improve LoRA acquisition, but it still trails full-model ER and can forget more of the capabilities learned in intermediate phases. Continued ER training further improves performance without additional teacher training or online data collection. Model merging and policy distillation can improve some results, but training additional teachers incurs substantial cost. We therefore recommend full-model ER as the preferred practical recipe for large-scale phase-wise VLA continual learning.

AI USE STATEMENT

We use generative AI to help refine the text and prepare figures, as well as to support discussions of experimental design and results, code and configuration checks, and experiment execution. The authors lead all aspects of the research, from experimental design and execution to result verification and interpretation, and take full responsibility for the paper's content and conclusions.

REPRODUCIBILITY STATEMENT

Sections 3–5 describe the compared methods, task partitions, and main training and evaluation settings. Our ER implementation follows established practice, with the configurations and training schedules specified in the paper. Appendix A provides additional details on replay, training schedules, evaluation, and diagnostic analyses. The code repository is being organized, and we will publicly release the code and experiment configurations as soon as it is ready.

REFERENCES

Rahaf Aljundi, Lucas Caccia, Eugene Belilovsky, Massimo Caccia, Min Lin, Laurent Charlin, and Tinne Tuytelaars. Online continual learning with maximally interfered retrieval, 2019a. URL https://arxiv.org/abs/1908.04742.

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning, 2019b. URL https://arxiv.org/abs/1903.08671.

Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, Cody Blakeney, and John P. Cunningham. Lora learns less and forgets less. *arXiv preprint arXiv:2405.09673*, 2024.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS 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, 2026. URL https://arxiv.org/abs/2606.03598.

Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning, 2020. URL https://arxiv.org/abs/2004.13513.

Arthur Douillard, Yifu Chen, Arnaud Dapogny, and Matthieu Cord. Plop: Learning without forgetting for continual semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4040–4050, June 2021.

Maximilian Du, Zhanyi Sun, Chen Xu, Paarth Shah, Masha Itkina, and Shuran Song. Memory anchors for continual robot learning, 2026. URL https://arxiv.org/abs/2608.26545.

Zhen Fang, Wenxuan Huang, Yu Zeng, Yiming Zhao, Shuang Chen, Kaituo Feng, Yunlong Lin, Lin Chen, Zehui Chen, Shaosheng Cao, et al. Flow-opd: On-policy distillation for flow matching models. *arXiv preprint arXiv:2605.08063*, 2026.

Antonio Andrea Gargiulo, Donato Crisostomi, Maria Sofia Bucarelli, Simone Scardapane, Fabrizio Silvestri, and Emanuele Rodolà. Task singular vectors: Reducing task interference in model merging, 2025. URL https://arxiv.org/abs/2412.00081.

Yao He, Gan Sun, Wenqi Liang, Fazeng Li, and Yang Cong. Towards human-like physical intelligence: Lifelong vision-language-action learning for robotic manipulation, 2026. URL https://arxiv.org/abs/2607.14852.

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. *arXiv preprint arXiv:2106.09685*, 2021.

Jiaheng Hu, Jay Shim, Chen Tang, Yoonchang Sung, Bo Liu, Peter Stone, and Roberto Martin-Martin. Simple recipe works: Vision-language-action models are natural continual learners with reinforcement learning, 2026. URL https://arxiv.org/abs/2603.11653.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

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, March 2017. ISSN 1091-6490. doi: 10.1073/pnas.1611835114. URL http://dx.doi.org/10.1073/pnas.1611835114.

Zhizhong Li and Derek Hoiem. Learning without forgetting, 2017. URL https://arxiv.org/abs/1606.09282.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling, 2023. URL https://arxiv.org/abs/2210.02747.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, qiang liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=xzEtNSuDJk.

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 *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=VzdSHEab4G.

Jie Liu, Gongye Liu, Jiajun Liang, Yangguang Li, Jiaheng Liu, Xintao Wang, Pengfei Wan, Di Zhang, and Wanli Ouyang. Flow-grpo: Training flow matching models via online rl, 2025. URL https://arxiv.org/abs/2505.05470.

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, 2024. URL https://arxiv.org/abs/2310.05905.

David Lopez-Paz and Marc' Aurelio Ranzato. Gradient episodic memory for continual learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/f87522788a2be2d171666752f97ddebb-Paper.pdf.

Kevin Lu and Thinking Machines Lab. On-policy distillation. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

Soroush Nasiriany, Sep Nasiriany, Abhiram Maddukuri, and Yuke Zhu. Robocasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *International Conference on Learning Representations*, volume 2026, pp. 98643–98667, 2026.

NVIDIA, Francesco Ferroni, Prithvijit Chattopadhyay, Greg Heinrich, Mike Ranzinger, Roberto Amoroso, Alice Luo, Andrew Wang, and Ming-Yu Liu. Cosmos-embed1: A joint video-text embedder for physical ai, 2025. URL https://research.nvidia.com/labs/dir/cosmos-embed1/.

Kai Nylund, Suchin Gururangan, and Noah Smith. Time is encoded in the weights of finetuned language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2571–2587, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.141. URL https://aclanthology.org/2024.acl-long.141/.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *2017 IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 5533–5542. IEEE, 2017.

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. ISSN 2377-3774. doi: 10.1109/lra.2026.3693992. URL http://dx.doi.org/10.1109/LRA.2026.3693992.

Reece Shuttleworth, Jacob Andreas, Antonio Torralba, and Pratyusha Sharma. Lora vs full fine-tuning: An illusion of equivalence. *arXiv preprint arXiv:2410.21228v3*, 2025.

George Stoica, Pratik Ramesh, Boglarka Ecsedi, Leshem Choshen, and Judy Hoffman. Model merging with svd to tie the knots, 2024. URL https://arxiv.org/abs/2410.19735.

Shahram Najam Syed, Yatharth Ahuja, Arthur Jakobsson, and Jeff Ichnowski. Expres-vla: Specializing vision-language-action models through experience replay and retrieval, 2026. URL https://arxiv.org/abs/2511.06202.

Weikang Wan, Yifeng Zhu, Rutav Shah, and Yuke Zhu. Lotus: Continual imitation learning for robot manipulation through unsupervised skill discovery, 2024. URL https://arxiv.org/abs/2311.02058.

Yuxuan Wu, Guangming Wang, Zhiheng Yang, Tianchen Deng, Maoqing Yao, Brian Sheil, and Hesheng Wang. Continually evolving skill knowledge in vision language action model, 2026. URL https://arxiv.org/abs/2511.18085.

Yajat Yadav, Zhiyuan Zhou, Andrew Wagenmaker, Karl Pertsch, and Sergey Levine. Robust fine-tuning of vision-language-action robot policies via parameter merging, 2026. URL https://arxiv.org/abs/2512.08333.

Chao Yu, Yuanqing Wang, Zhen Guo, Hao Lin, Si Xu, Hongzhi Zang, Quanlu Zhang, Yongji Wu, Chunyang Zhu, Junhao Hu, Zixiao Huang, Mingjie Wei, Yuqing Xie, Ke Yang, Bo Dai, Zhexuan Xu, Jiakun Du, Xiangyuan Wang, Xu Fu, Letong Shi, Zhihao Liu, Kang Chen, Weilin Liu, Gang Liu, Boxun Li, Jianlei Yang, Zhi Yang, Guohao Dai, and Yu Wang. Rlinf: Flexible and efficient large-scale reinforcement learning via macro-to-micro flow transformation, 2025. URL https://arxiv.org/abs/2509.15965.

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

Zhide Zhong, Haodong Yan, Junfeng Li, Junjie He, Tianran Zhang, and Haoang Li. Vla-opd: Bridging offline sft and online rl for vision-language-action models via on-policy distillation, 2026. URL https://arxiv.org/abs/2603.26666.

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

A Implementation Details

A.1 Data and Task Protocols

LIBERO task composition. Table 3 gives the B0–B9 task numbering used throughout the paper. For the task-wise/phase-wise and composition comparisons, we first fine-tune the pretrained $\pi_{0.5}$ model on B0–B4. Each subsequent run starts independently from this checkpoint and learns its designated subset of B5–B9 without replay. The three multi-task subsets are B5+B6+B7+B8, B5+B6+B8+B9, and B5–B9; the single-task runs learn B5 through B9 separately. Both the initial B0–B4 training and the continual updates use 3,000 optimizer steps, minibatch size 128, and AdamW with gradient-norm clipping at 1. The learning rate warms up for 1,000 steps to 1.77×10^{-5}, then follows cosine decay to 10^{-6} by step 3,000. We initialize a fresh optimizer for each run and do not use an exponential moving average of model parameters.

Table 3: LIBERO-10 task identifiers used in the paper.

ID	Task instruction
B0	Put both the alphabet soup and the tomato sauce in the basket.
B1	Put both the cream cheese box and the butter in the basket.
B2	Turn on the stove and put the moka pot on it.
B3	Put the black bowl in the bottom drawer of the cabinet and close it.
B4	Put the white mug on the left plate and put the yellow and white mug on the right plate.
B5	Pick up the book and place it in the back compartment of the caddy.
B6	Put the white mug on the plate and put the chocolate pudding to the right of the plate.
B7	Put both the alphabet soup and the cream cheese box in the basket.
B8	Put both moka pots on the stove.
B9	Put the yellow and white mug in the microwave and close it.

LIBERO update granularity. Figure 6b compares full-parameter SFT without replay from LIBERO-90 to LIBERO-10. Every run starts independently from the same LIBERO-90 checkpoint at step 1,109, whose old- and new-task success rates are 96.77% and 12.50%, respectively. We order the ten new tasks as B0 through B9 and partition this sequence into contiguous groups of one, two, five, or ten tasks. The respective group budgets are 50, 100, 250, and 500 steps; all four runs use 500 total steps with global batch size 128.

For each new task, we draw 6,400 training sample indices with replacement and reuse the same per-task draws in every run, totaling 64,000 sample presentations per run. We shuffle once within each group, concatenate the groups in order, and use one continuous dataloader. Model parameters, AdamW state, and the learning-rate schedule continue across group boundaries without restarting. The learning rate is constant at 3.5355×10^{-5}, the gradient-norm clipping threshold is 1.0, and the random seed is 42. All model parameters are trainable; we use neither LoRA nor an exponential moving average. Each run uses eight GPUs with FSDP on a single node. Old-task and new-task success are reported separately after the complete sequence.

LIBERO task scale. This experiment uses a different starting model from the B0–B4 experiments: $\pi_{0.5}$ is first trained on all available tasks of LIBERO-90, and all ten LIBERO-10 tasks are new. The initial training uses 1,109 updates with minibatch size 512, approximately one pass over the old-task data. Its learning rate warms up for 100 steps to 5×10^{-5} and decays to 5×10^{-6}. Observation and action normalization statistics are computed from the old-task data and reused during continual learning.

At each task scale, ER and ER with LoRA start from this same old-task checkpoint and use the same task groups and replay subset. Each group is trained independently for 100 updates with a fresh optimizer and minibatch size 64 (20 old and 44 new samples). The learning rate warms up for 10 steps to 5×10^{-5}, then decays to 5×10^{-6}. Two-task groups are {B5,B6}, {B0,B7}, {B2,B3}, {B4,B9}, and {B1,B8}; five-task groups are {B0,B3,B5,B6,B7} and {B1,B2,B4,B8,B9}. These partitions are fixed before training using seed 42. The single-task and ten-task settings cover each task separately and all ten tasks jointly, respectively. We average the runs equally within each scale. Thus, each task appears once per scale, while the total update budget per run stays fixed.

RoboCasa365. We follow the benchmark's lifelong-learning partition: 65 tasks in Phase1 and 20 new tasks in each of Phases 2–4. Training uses the corresponding human demonstrations, with task instructions supplied as language prompts. The Phase1 base is trained for 25k steps; the separate teachers for Phases 2–4 are each trained for 15k steps from the pretrained $\pi_{0.5}$ checkpoint, using only

their own phase's demonstrations. Continual students instead inherit the model from the preceding update. We keep a shared observation/action normalization across the RoboCasa models. The policy predicts 32-action chunks on RoboCasa and 10-action chunks on LIBERO, using the benchmark-specific input and action transforms.

RoboCasa update granularity. The comparison in Figure 6a starts from the Phase1 model and uses only Phase2 demonstrations, without replay. The 20 new tasks are partitioned into 20, 4, 2, or 1 disjoint groups of 1, 5, 10, or 20 tasks, respectively. Groups are trained sequentially, carrying the learned model parameters into the next group and using only the current group's data. Each group of g tasks receives $50g$ optimizer steps with batch size 128. Every setting therefore uses 1,000 steps in total, with the same average training-sample budget per task. Phase1 and Phase2 success are reported separately after the complete sequence.

A.2 ER and Continued Training

Replay construction and sampling. For RoboCasa ER, we select a fixed subset of demonstration frame indices from each old task, amounting to 2% of its data. Each index supplies an observation and its associated action chunk. The selected indices remain fixed during an update; when a new phase becomes old, its data become eligible for replay in subsequent updates. Sampling within the old and new pools is proportional to their numbers of available frames. This is distinct from allocating equal sampling probability to every task.
The stored-data fraction and minibatch composition serve different purposes. The 2% fraction determines memory size, whereas the old/new minibatch split controls how often that memory is replayed. The main RoboCasa experiments use 160 old and 352 new samples per optimizer update, for a total of 512; Appendix B.7 specifies the smaller training budget used in the LoRA ablations. Both branches use the demonstration flow-matching loss in Equation 3. Full-model ER updates all parameters; sequential SFT uses only the new demonstrations.

Training schedule. Table 4 summarizes the main ER schedule comparison. ER15k warms up to a peak learning rate of approximately 3.54×10^{-5}, holds that rate, and applies cosine decay over the final 500 steps. ER30k extends the high-learning-rate portion rather than adding low-learning-rate continuation. In the Phase1-to-Phase2 comparison, the two schedules share training through step 14,500, after which the 15k branch decays while the 30k branch continues. We use AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, weight decay 10^{-10}, and gradient-norm clipping at 1.

Table 4: RoboCasa ER training schedules. The continuation column reports additional training from ER15k for the no-teacher comparison in Figure 7.

Setting	ER15k	ER30k	ER15k + continuation
Additional updates	15,000	30,000	3,000 or 5,000
Old/new minibatch	160/352	160/352	160/352
Warmup steps	1,000	1,000	100
Constant-rate steps	13,500	28,500	2,900 or 4,900
Cosine-decay steps	500	500	0
Peak/constant learning rate	3.54×10^{-5}	3.54×10^{-5}	5×10^{-6}
Learning rate after decay	10^{-6}	10^{-6}	—

Low-learning-rate continuation. The plain ER+3k run loads the parameters of ER15k, initializes a fresh optimizer once, and performs another 3,000 ER updates. The learning rate increases linearly to 5×10^{-6} over the first 100 continuation steps and remains constant thereafter. It reuses the original ER replay subset and the full current-phase demonstration set. ER+5k extends this continuation to a total of 5,000 extra steps, preserving the optimizer state at the 3k boundary. In the full-stream ER+3k experiment, each continual phase receives 15k ER steps followed by 3k additional ER steps; the resulting model is passed to the next phase. No teacher or on-policy buffer is used in this stream.

A.3 Compared Methods

LoRA. The RoboCasa LoRA comparisons use rank 128 in the language model and rank 256 in the action expert. The LIBERO task-scale comparison uses the rank-16 language-model and rank-32 action-expert adapters. We use OpenPI's LoRA variants for the attention and feed-forward projections: the language-model and action-expert base weights are frozen, while the adapters, image encoder, action input/output projections, and timestep MLP are trainable. The main RoboCasa

ER+LoRA 15k configurations use the same 160/352 replay split as full-model ER15k. The $1\times$ configuration uses the reference schedule in Table 4. In the $5\times$ configuration, only the LoRA adapter parameters use a fivefold learning-rate multiplier; the image encoder, action input/output projections, and timestep MLP remain trainable with the unchanged reference schedule. In the multi-phase stream, Phase1 uses full-parameter SFT. After each subsequent update, we merge the trained adapters into the base weights and initialize fresh adapters for the next phase. The trainable modules outside the adapters listed above remain trainable throughout.

Model merge. Dense merge interpolates the pre-update student and the current-phase teacher using the phase-based weighting in Equation 4. We either evaluate this interpolation directly or use it to initialize 15k full-model ER updates. SVD merge uses rank 128 and the same phase-based weighting; the merged initialization is then trained with full-model ER, not LoRA-only optimization. RETAIN interpolates the pre-update student and the corresponding ER15k model with 20% and 80% weights, respectively, and is evaluated without further training. Appendix B.6 reports the full coefficient sweep, including the equal-weight configuration. In the multi-phase teacher-assisted stream, each phase applies SVD merge, ER, and IPD in sequence. Consequently, a pre-IPD checkpoint omits only the current phase's IPD update and still inherits IPD performed in earlier phases.

Rollout-based updates. For the refinement comparison in Figure 7, Flow-GRPO, Flow-OPD, and IPD start from the same SVD+ER15k model. Flow-OPD and IPD use the frozen Phase2 teacher for velocity-field supervision; Flow-GRPO uses task-success rewards. The direct Flow-OPD and Flow-GRPO comparisons in Figure 2 instead start from the Phase1 base. Old replay retains its demonstration flow-matching objective in these methods, while Flow-OPD and IPD share the same new-task distillation loss and Flow-GRPO uses a reward-based objective.
The SVD+ER refinement runs collect 128 student episodes per Phase2 task using the RoboCasa v1.0.1 task-specific horizons and the `pretrain` environment split. The student executes its full 32-action chunk before replanning. Stochastic flow sampling uses ten denoising steps and noise level 0.5. Flow-GRPO and Flow-OPD collect 16 episodes per task in each of eight rounds, updating the student between rounds. Their hybrid optimization uses 352 rollout samples and 160 replay samples per minibatch, one training epoch over each round's rollout data, and learning rate 5×10^{-6}.
Flow-GRPO uses groups of eight trajectories, a clipped policy objective with clipping range $[0.8, 1.2]$, and binary success rewards. Groups with all successes or all failures are excluded from its policy-gradient update. Flow-OPD instead evaluates the teacher velocity field at the student's sampled noisy actions and minimizes the time-weighted velocity-field MSE in Equation 7; it does not filter trajectories by reward. IPD collects its rollout buffer before offline optimization and reuses it for 1,000 updates with the same time-weighted velocity-field MSE as Flow-OPD in Equation 7. It uses a fresh optimizer, a 100-step linear warmup to 5×10^{-6}, and a constant learning rate thereafter. The three refinement methods share a fixed old-task replay subset reserved separately from the initial ER subset.

A.4 Evaluation

We measure success using each benchmark's task-completion criterion. Old- and new-task success are reported separately. Within a RoboCasa phase, we average task success rates equally; we do not average success across phases.
For LIBERO task-wise/phase-wise and composition experiments, we evaluate each task on the 50 official initial states with indices 0–49, evaluation seed 7, ten settling steps, and replanning every five executed actions. The maximum policy length is 520 steps. The task-scale experiment uses four initial states per old task and eight per new task, with maximum policy lengths of 400 and 520 steps, respectively. Both methods use the same states within each scale comparison. These scale results average task groups, not repeated random seeds.
RoboCasa main experiments use 128 evaluation episodes per task by default. Evaluation uses the `pretrain` split and official v1.0.1 task-specific horizons. The 2k-step task-mixture analysis uses 16 episodes per old task. The 20-task retention screen uses four episodes per old task after each 1k-step single-task update. Evaluation iterates through an ordered reset-seed sequence, rather than repeatedly evaluating the same initial state. Final checkpoint evaluations are separate from training-rollout success statistics.

A.4.1 Evaluation Confidence Intervals

Table 2 and Figure 7 report 95% confidence intervals for task-averaged success rates. For a fixed set of M tasks, let s_j be the number of successful episodes among n_j evaluation episodes for task

j, and let $\hat{p}_j = s_j/n_j$. We average tasks equally, $\bar{p} = M^{-1}\sum_j \hat{p}_j$, and compute the standard error under independent evaluation episodes:

$$\widehat{\mathrm{SE}}(\bar{p}) = \frac{1}{M}\sqrt{\sum_{j=1}^{M}\frac{\hat{p}_j(1-\hat{p}_j)}{n_j}}, \qquad \mathrm{CI}_{95\%} = \bar{p} \pm 1.96\widehat{\mathrm{SE}}(\bar{p}). \tag{8}$$

We use one training seed per configuration for the RoboCasa phase-wise experiments because of their computational cost. A single Phase1-to-Phase2 training run requires more than 10 hours on 32 NVIDIA A100 GPUs, excluding evaluation. The full four-phase sequence requires three continual updates and approximately three times the training compute, with additional evaluation after each phase. Repeating these experiments with three training seeds is beyond our available computational budget. Reporting a single training seed under computational constraints also has precedent in language-model research; for example, Nylund et al. (2024) use one fine-tuning seed for this reason. These normal-approximation intervals therefore quantify evaluation variability for a fixed trained policy and task set, rather than variation across training seeds. The approximation can underestimate uncertainty when a task has no failures or no successes. Success rates are reported in percent and interval half-widths in percentage points. Displayed results are rounded to one decimal place; interval calculations use unrounded task-level proportions.

A.5 Transfer Metrics

We report backward transfer (BWT) and forward transfer (FWT), adapting the continual learning metrics of GEM to phase-wise updates (Lopez-Paz & Ranzato, 2017).

Backward transfer. Let $R_{k,j}$ denote the mean task success rate, in percent, on phase j after completing phase k, and let n_j denote the number of tasks in phase j. We average changes equally over old tasks:

$$\mathrm{BWT}_k = \frac{\sum_{j<k} n_j(R_{k,j} - R_{j,j})}{\sum_{j<k} n_j}. \tag{9}$$

Each reference $R_{j,j}$ is measured immediately after the same method learns phase j. Negative BWT indicates forgetting; positive BWT indicates improved old-task performance. For one update, BWT reduces to the change in old-task success. The LIBERO task-scale experiment (Appendix B.2) averages these changes over independently trained task groups.

Forward transfer. For a new task τ, we measure Phase1-to-Phase2 FWT as

$$\mathrm{FWT}_{1\to 2}(\tau) = S(\pi_1, \tau) - S(\pi_{\mathrm{pretrained}}, \tau), \tag{10}$$

where S is success in percent, π_1 is the Phase1 policy, and $\pi_{\mathrm{pretrained}}$ is the original pretrained $\pi_{0.5}$ policy. We average equally over new tasks. Both policies are evaluated before Phase2 training, so this FWT is shared by methods initialized from the same Phase1 policy. We report only this transition, not a full-stream FWT average. Both transfer metrics use percentage points (pp).
RoboCasa FWT uses 64 episodes per task and policy with matched reset seeds and physical initial states, task-specific horizons, action chunks of 32, and ten denoising steps. LIBERO uses 50 official initial states per task and policy, 520 policy steps plus ten settling steps, action chunks of ten, replanning every five steps, and ten denoising steps. Each policy pair uses the same normalization.

A.6 Diagnostic Analyses

Visual and VLA features. The visual proxy uses frozen Cosmos-Embed1-448p video features (NVIDIA et al., 2025). For each demonstration camera view, we uniformly sample 16 frames, form two interleaved eight-frame clips, and average their normalized embeddings. We then combine the camera views into an episode representation. Task-pair similarity is the mean of the highest 5% of cross-episode cosine similarities. The VLA proxy instead uses the frozen Phase1 policy: we pool valid image-encoder tokens across cameras, average the observation features within each task, and compute cosine similarity between task centroids.

Gradient and loss probes. Gradient similarity is the cosine between demonstration flow-matching gradients of the frozen Phase1 model, using all model parameters. For the loss probe, two copies of that model are independently fine-tuned on Kettle and TurnOff for 64 steps without replay. We measure flow-matching loss on a fixed set of 16 demonstration windows per old task, using the same

noise realizations before and after each update. The predictor is the increase after TurnOff training minus the increase after Kettle training. Thus, larger predictor values indicate less old-task loss damage under Kettle.

In Figure 5, all panels use the same response: the per-old-task difference in retained successes between Kettle and TurnOff training. Visual and VLA predictors are Kettle-minus-TurnOff similarities, whereas the gradient predictor is cosine similarity with Kettle. Lines are ordinary least-squares fits and Pearson's r measures linear association. For the full-task analysis in Appendix B.3, we average each candidate task's similarity to the 65 old tasks and relate this score to retention after single-task training. We report Spearman's rank correlation with average ranks for ties and two-sided permutation tests using 100,000 permutations of the 20 task-level responses. The within-candidate analysis correlates similarity with per-old-task success change and summarizes the 20 candidate correlations by their median.

The motivation, procedures, and implementation of the LIBERO anchor-based screens are described together in Appendix B.3.

B Additional Results

B.1 Backward and Forward Transfer

Metric definitions and evaluation settings are given in Appendix A.5.

Backward transfer across 125 tasks. Table 5 weights previous phases by their task counts (65, 20, and 20 when evaluating after Phase4). Values use reported rates in Table 2 and Appendix B.8 and are approximate; LoRA $5\times$ BWT is computed before rounding rates for display. The SVD+ER+IPD row uses the post-IPD result at each phase. Pre-IPD checkpoints are not an independent IPD-free stream. ER and ER+3k have small net changes. LoRA at $1\times$ has favorable BWT but weak acquisition; at $5\times$, acquisition improves while BWT becomes negative after further updates. BWT measures preservation relative to each method's own acquired performance, not the absolute level of that performance.

Table 5: Approximate task-averaged BWT (pp) across RoboCasa365 phases, calculated from reported rates in Table 2 and Appendix B.8.

Method	After Phase2	After Phase3	After Phase4
ER	$+0.2$	$+0.8$	-0.6
ER+3k	-0.1	-0.7	-0.5
ER+LoRA ($1\times$)	$+1.6$	$+2.4$	$+0.5$
ER+LoRA ($5\times$)	$+1.7$	-1.1	-2.7
SVD+ER+IPD	-0.2	$+0.4$	-2.4

Backward transfer at different task scales. The LIBERO task-scale study (Appendix B.2) starts from a different checkpoint than the B0–B4 study. Table 6 subtracts the shared starting checkpoint's old-task success ($347/360$, approximately 96.39%) from each group's old-task success and averages groups equally within a scale. BWT is calculated from unrounded success rates and then rounded to one decimal place. Each group is an independent update from the same base; the columns do not represent successive updates along one stream.

Table 6: BWT (pp) for the LIBERO task-scale study.

New tasks	1	2	5	10
ER	-7.6	-8.1	-11.1	-10.6
ER+LoRA	-1.5	-2.6	-4.2	-8.3

Phase1-to-Phase2 forward transfer. Table 7 summarizes FWT for the single-task and phase-wise settings; Table 8 reports individual tasks.

Relationship to retention. LIBERO's gain is concentrated entirely on B7 (Table 8), consistent with its support role, but learning B0–B4 jointly does not isolate which old tasks cause this transfer. On RoboCasa, Clear and Kettle have zero measured FWT, while TurnOff has positive FWT despite poor old-task retention. Across the 20 tasks, FWT and old-task success after the 1k-step single-task updates in Appendix B.3 have Pearson correlation 0.047 and Spearman correlation 0.160. These descriptive associations are weak. Single-task retention is not the marginal effect of adding a task

Table 7: Phase1-to-Phase2 FWT. FWT is in percentage points. Single-task rows are subsets of their benchmark's full new-task set.

Benchmark	New-task set	FWT
RoboCasa365	TurnOff	+3.1
RoboCasa365	All 20 Phase2 tasks	+1.3
LIBERO-10	B5	0.0
LIBERO-10	B5–B9	+8.4

to a mixture, and zero measured success does not exclude partial skill transfer. These results do not establish FWT as a reliable proxy for retention-support effects.

Table 8: Per-task forward transfer. RoboCasa uses 64 episodes per policy per task; LIBERO uses 50. FWT is in percentage points.

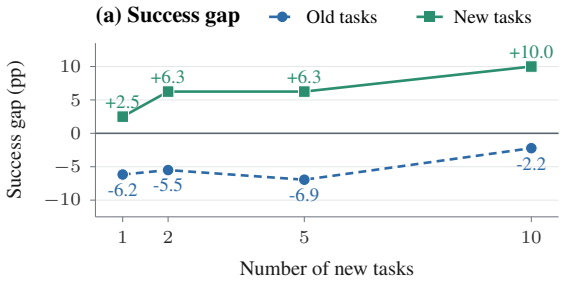
Task	FWT
RoboCasa365	
KettleBoiling	0.0
PrepareCoffee	0.0
ClearCuttingBoard	0.0
RinseSinkBasin	0.0
LoadDishwasher	0.0
WashLettuce	0.0
StrainerSetup	+1.6
PrepareToast	0.0
ChooseMeasuringCup	0.0
OrganizeCleaningSupplies	0.0
StackBowlsCabinet	+6.3
RecycleStackedYogurt	+1.6
AddMarshmallow	+3.1
ScrubCuttingBoard	+3.1
TurnOffSimmeredSauceHeat	+3.1
SetUpCuttingStation	0.0
VeggieDipPrep	+3.1
PreSoakPan	0.0
OrganizeCondiments	+4.7
PrepForSanitizing	0.0
LIBERO-10	
B5	0.0
B6	0.0
B7	+42.0
B8	0.0
B9	0.0

B.2 Task Scale on LIBERO

We examine scale-dependent learning dynamics with a larger old-task pool: a shared model trained on LIBERO-90 learns new tasks from LIBERO-10 using ER or ER with LoRA. For LIBERO's small, simple new-task set, we use LoRA ranks of 16 for the language model and 32 for the action expert. To reduce the influence of task composition, we train on each of the ten new tasks separately and average the results for the single-task setting. For two-task updates, we divide the ten tasks into five disjoint pairs and average the five runs. Similarly, five-task and ten-task updates use two groups and one group, respectively. This scale comparison uses 100 optimizer steps per run.

From one to ten new tasks, the retention gap between ER and ER with LoRA narrows, while the acquisition gap widens in favor of ER (Figure 8). BWT results show the same retention advantage for LoRA (Appendix B.1). Learning more new tasks together may bring in more support tasks, but the central observation is that task scale changes the learning dynamics: the balance between acquiring new capabilities and retaining old ones shifts as more tasks are learned together.

At 1,000 steps, both methods improve on old and new tasks, while the acquisition gap narrows from 10 to 5 pp. With each method reaching 100% success on six tasks, LIBERO-10 offers limited discrimination of acquisition performance; on the more challenging B9 task, however, ER still outperforms ER+LoRA by 25 pp (62.5% vs. 37.5%).



(b) Success rate (%)

New tasks	Old tasks ER	Old tasks +LoRA	New tasks ER	New tasks +LoRA
1	88.8	**94.9**	**85.0**	82.5
2	88.3	**93.8**	**91.3**	85.0
5	85.3	**92.2**	**81.3**	75.0
10	85.8	**88.1**	**70.0**	60.0
10*	92.5	**93.6**	**92.5**	87.5

*1,000 steps; other rows: 100 steps

Figure 8: Task scale and VLA continual learning dynamics on LIBERO. (a) Success-rate differences in percentage points; positive values favor full-model ER over ER with LoRA. (b) Absolute success rates. Results average 10, 5, 2, and 1 disjoint groups for 1, 2, 5, and 10 new tasks, respectively. The starred row reports 1,000-step results.

B.3 Semantic Analysis

Full-task analysis on RoboCasa365. The pairwise analysis in Figure 5 examines which old tasks benefit more from Kettle than TurnOff. Here we test whether similarity can instead identify promising new-task candidates among all 20 RoboCasa365 Phase2 tasks. Each task is trained alone for 1k steps and evaluated over the 65 Phase1 tasks. Figure 9 compares its Phase1 success rate with its mean visual, gradient, and VLA vision-feature similarity to the old-task pool.

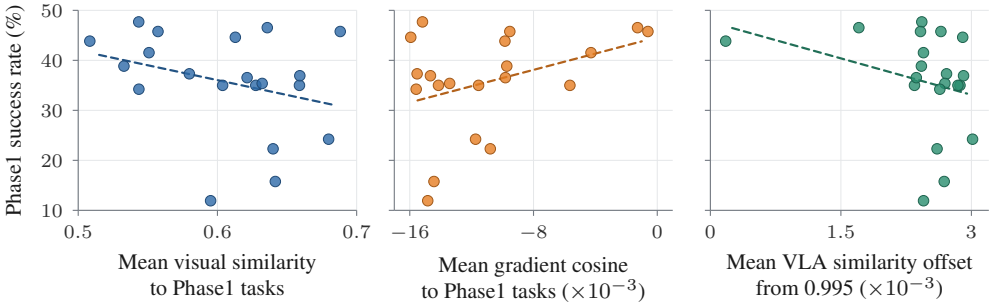


Figure 9: Full-task analysis of low-cost retention proxies. Each point is one of the 20 Phase2 tasks. The horizontal axes show its mean similarity to the 65 Phase1 tasks, and the vertical axis shows Phase1 success after single-task training. Dashed lines are linear fits.

Across all 20 tasks, visual, gradient, and VLA similarity have candidate-level Spearman correlations of -0.30, 0.31, and -0.35, respectively; none is significant ($p \geq 0.13$). In the same order, the median within-candidate correlations between similarity and per-task retention are -0.01, -0.03, and -0.05. These trends do not provide a consistent screening signal. Together with the pairwise diagnostic, the results show that these semantic analyses do not establish a high-confidence relationship for support-task identification. This broader screen measures single-task retention, not the effect of adding a task to a mixture.

Memory-anchor-based analysis on LIBERO-10. The preceding correlation analyses do not provide a consistent signal for identifying support tasks. We therefore investigate whether a more targeted analysis of local observation overlap and action disagreement can better characterize task relationships. We adapt the recent Memory Anchors framework (Du et al., 2026), which identifies old-task examples that are important for retention under experience replay. The method locates new observations close to old observations in the policy's representation space, selects the new samples with the largest disagreement between the old policy and demonstration actions, and retrieves nearby old samples as anchors. Enriching replay with these anchors is intended to preserve old behavior where similar observations require conflicting actions.

We adapt this procedure to screen new-task candidates on LIBERO-10 using the frozen B0–B4 policy. We consider two possible indicators of support: observations whose demonstrated actions are already compatible with the old policy, and observations close to a conflict region induced by another new task. The first screen selects low-error samples from the B5–B9 overlap with old data; the second retrieves B6–B9 samples near high-error observations from B5. Both are offline analyses of demonstration data, without updating the reference policy or training Anchor ER.

We represent each observation using input tokens before the frozen policy's language-model backbone. We separately mean-pool each valid camera's tokens and the instruction tokens, normalize

the three feature blocks, and concatenate them. Smaller Euclidean distance means greater similarity in this representation. For each new observation, we summarize its distances to old observations by the minimum and the 1st, 5th, and 10th percentiles. We apply k-means with two clusters to these four distance statistics. We use random seed 42 and run the clustering from 20 different centroid initializations, retaining the solution with the smallest within-cluster sum of squared distances. The cluster with lower mean distances defines the *overlap subset*.

For action error, we perturb a normalized demonstration chunk to flow time $t = 0.2$, let the frozen policy reconstruct it using two Euler steps of size -0.1, and measure Euclidean reconstruction error over ten actions and seven valid action dimensions. We average over four fixed noise draws. Each evaluated observation must have the full action sequence required by the reconstruction probe; observations near a demonstration's end that lack this sequence are excluded. Low error indicates compatibility with the demonstrated action under this probe; high error is a proxy for disagreement.

For the first screen, we pool B5–B9 observations, identify their overlap with B0–B4, and retain the lowest-error 10% of that pooled overlap. Such samples might rehearse behavior already supported by the old policy. This reverses the high-error selection used to locate conflict in Memory Anchors. Figure 10(a) reports each task's overlap and selected low-error counts, both divided by its number of valid action windows. Because selection is pooled across tasks, the selected fraction need not equal 10% within each task.

For the second screen, we treat B5 as the incoming new task and B6–B9 as a surrogate "old-task" data pool from which to retrieve anchors. This designation specifies their role in retrieval only: the reference policy has been trained on B0–B4, not B6–B9, and remains frozen throughout. We first find B5 observations overlapping with the actual old-task data B0–B4 and select the highest-error 10% as *conflict queries*, where the reference policy struggles to reconstruct B5's demonstrated actions. For each B6–B9 observation, we find the five nearest conflict queries and use the median of their distances as its score. We compute the same scores for B0–B4 observations and set a common cutoff at their 10th percentile. Candidate observations below this cutoff are retrieved. Figure 10(b) divides their count by all observations in each candidate task; B5 supplies the queries and is not ranked. We also repeat retrieval with one and ten neighbors.

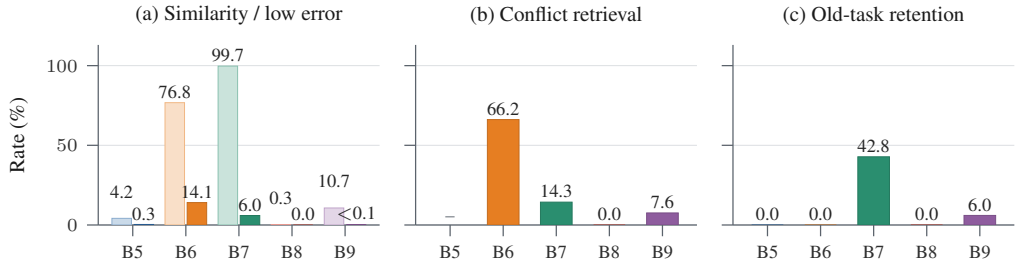


Figure 10: LIBERO-10 screening. (a) Overlap (pale) and low-error candidates (solid), divided by valid action windows. (b) B5-conditioned retrieval, divided by all observations; B5 is not a candidate. (c) B0–B4 success after single-task training for 3k steps (Figure 3).

In Figure 10(a), B7 has the largest overlap fraction (99.7%), and its low-error candidates have nearest matches in B0 or B1. However, the final low-error fraction is larger for B6 (14.1%) than B7 (6.0%). Screen 2 also ranks B6 above B7 (66.2% versus 14.3%), including with one or ten neighbors. This disagrees with panel (c): B7-only training retains 42.8% old-task success, whereas B6-only training retains 0%. The screening rankings thus differ from the observed single-task retention rankings. Memory Anchors was designed to select old-task examples for replay in relation to an incoming new task, whereas our analysis seeks new tasks that may preserve previously learned behavior. In particular, the surrogate B6–B9 pool contains tasks the reference policy has never learned. This difference in objective and data roles may limit the method's applicability to support-task identification. Moreover, panel (c) does not measure the effect of adding a candidate to B5. These results therefore concern our adapted screens and do not establish a limitation of Anchor ER for its original replay objective.

B.4 Effect of SVD Merge Rank

We ablate the truncation rank of SVD merge in two RoboCasa task settings, holding the student and teacher checkpoints fixed within each setting. We apply Equation 5 to the attention and feed-forward projection matrices, using rank r for the vision-language backbone and $2r$ for the action expert, capped by the matrix dimensions.

The first setting combines ten old tasks with five new tasks using student:teacher weights of $2:1$; the second combines five old tasks with five new tasks using equal weights. The second setting's old tasks are SearingMeat, StoreLeftoversInBowl, KettleBoiling, LoadDishwasher, and PrepareCoffee, and its new tasks are PackIdenticalLunches, PreSoakPan, RinseSinkBasin, ScrubCuttingBoard, and SetUpCuttingStation. The first setting uses all ten of these as old tasks and adds TongBuffetSetup, StockingBreakfastFoods, SweetenCoffee, CutBuffetPizza, and DeliverBrewedCoffee. Every plotted task is evaluated over 256 episodes. Overall success averages all tasks equally, giving old:new weights of $2:1$ and $1:1$ in the respective panels.

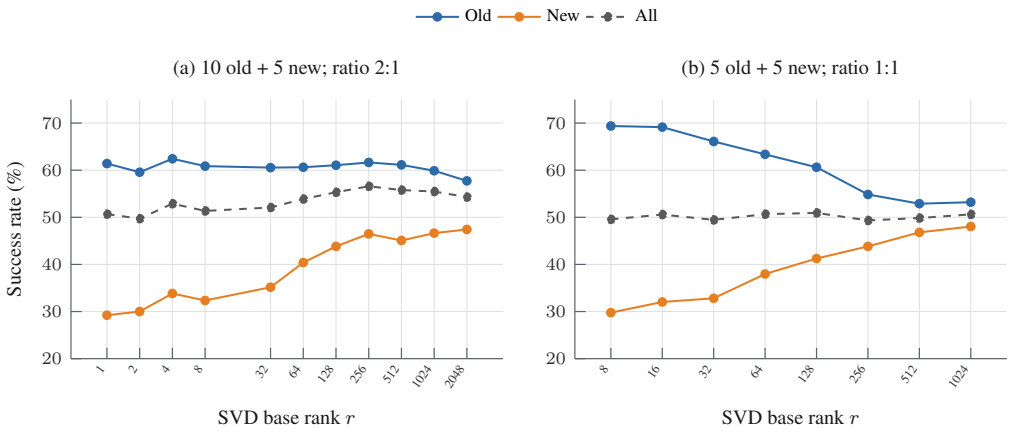


Figure 11: Rank ablation for SVD merge, evaluated without subsequent training. Curves report old-task, new-task, and overall success. Ratios denote student:teacher merge weights. Each reported rank has paired old- and new-task evaluations with 256 episodes per task.

Figure 11 shows that greater rank generally favors acquisition, but can reduce retention. In the ten-old/five-new setting, the highest observed overall success occurs at $r = 256$ (56.59%; old 61.64%, new 46.48%). Raising rank to 2048 increases new-task success to 47.42%, while old-task success falls to 57.73% and overall success to 54.30%. In the five-old/five-new setting, new-task success rises from 29.77% at $r = 8$ to 48.05% at $r = 1024$, whereas old-task success falls from 69.38% to 53.20%. The highest observed overall success is 50.94% at $r = 128$, compared with 50.63% at $r = 1024$.

These results support our choice of $r = 128$: it achieves strong overall performance in both settings (55.31% and 50.94%) while balancing old-task retention and new-task acquisition. Larger ranks can improve acquisition at the expense of retention, so maximizing rank does not consistently improve overall success.

B.5 Adapting Task-Wise VLA Continual Learning Methods

We further investigate how VLA continual learning methods developed for task-wise updates perform in our phase-wise setting, where multiple new tasks are introduced together. Many existing methods rely on specialized model architectures (e.g., Wu et al., 2026), while others require additional information (e.g., Chen et al., 2026). We therefore select Info-VLA (Zhao et al., 2026) and LifelongVLA (He et al., 2026), which can be applied to our $\pi_{0.5}$ backbone without requiring additional information, and adapt them with reference to their published designs and hyperparameter settings.

We first examine these adaptations on LIBERO-10 by adding B5 to a policy trained on B0–B4. As shown in Table 9, both methods achieve high new-task success while retaining substantially more old-task performance than sequential SFT, demonstrating effective continual learning in this single-task update. We then evaluate them on the RoboCasa365 Phase1-to-Phase2 transition, which introduces 20 new tasks simultaneously. Both methods preserve substantial old-task performance, but their new-task success rates are only 1.4% and 4.0%, respectively, compared with 35.39% for full-model ER. Thus, their strong performance in the single-task check does not directly translate into effective acquisition in this phase-wise setting.

One possible explanation is that the balance between retention and acquisition changes when many diverse tasks arrive together. LifelongVLA limits adaptation to low-rank branches, whereas Info-VLA freezes the vision-language encoder and regularizes the action expert against the preceding policy. These constraints may help preserve existing capabilities during single-task updates but

Table 9: Success rates (%) for VLA CL adaptations in task-wise and phase-wise updates.

Method	Old tasks	New tasks
LIBERO-10: B0–B4 → B5		
Sequential SFT	0.0	100.0
Info-VLA	87.6	98.0
LifelongVLA	95.6	94.0
RoboCasa365: Phase1 → Phase2		
Full-model ER	67.3	35.4
Info-VLA	64.3	1.4
LifelongVLA	62.8	4.0

become more restrictive when substantial adaptation is needed. This is a hypothesis rather than an established cause, since task domain and difficulty also differ between the two experiments.
These results should not be interpreted as evidence that either method is generally ineffective. We have not extensively tuned these adaptations for RoboCasa, and suitable configurations may improve their performance. However, such adaptation introduces additional choices, including the auxiliary-loss weights and contrastive temperature in Info-VLA, and the adapter rank, distillation weight, and relative learning rates in LifelongVLA. Searching these settings can become costly as the task collection grows, particularly if they need to be revisited for subsequent updates. In contrast, full-model ER achieves strong performance with our standard training configuration, without a method-specific hyperparameter search for this comparison. These observations suggest that methods developed for task-wise learning may require additional adaptation for phase-wise updates, while supporting simple ER as an effective practical starting point with a lower tuning burden.

B.6 Sensitivity to the RETAIN Mixing Weight

We sweep the ER checkpoint weight to examine how RETAIN balances old-task retention and new-task acquisition. For the RoboCasa365 Phase1-to-Phase2 transition, let θ_{pre} denote the pre-update checkpoint and θ_{ER} the ER-updated checkpoint. We interpolate the same two checkpoints as

$$\theta_\lambda = (1-\lambda)\theta_{\text{pre}} + \lambda\theta_{\text{ER}}, \qquad \lambda \in \{0, 0.1, \ldots, 1\}. \tag{11}$$

Here λ is the ER checkpoint weight: $\lambda = 0$ recovers the pre-update policy, $\lambda = 0.5$ gives the equal-weight merge, and $\lambda = 1$ recovers ER without merging. Each policy is evaluated without further training on 65 old tasks and 20 new tasks, using 520 old-task episodes and 160 new-task episodes. We report the two task groups separately because they differ in task composition and difficulty. The main comparison uses 80% ER weight; the sweep also includes the equal-weight configuration.

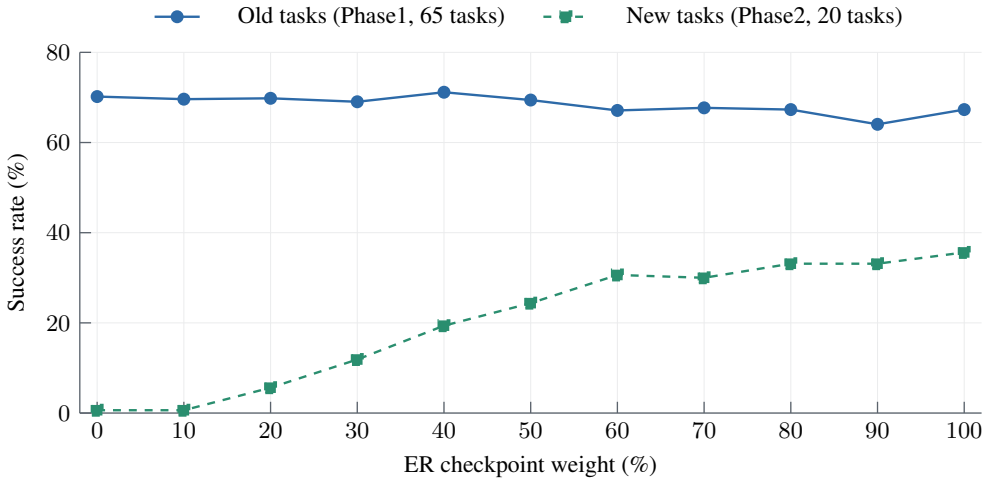


Figure 12: RETAIN mixing-weight sweep on RoboCasa365 Phase1-to-Phase2. The horizontal axis gives the ER checkpoint weight; the remaining weight is assigned to the pre-update checkpoint. Solid circles show old-task success and dashed squares show new-task success. All 11 coefficients are evaluated with 520 old-task and 160 new-task episodes each. Results are reported separately for the two task groups.

Figure 12 shows an overall increase in new-task acquisition as more weight is assigned to ER, accompanied by a comparatively modest decrease in old-task success, with non-monotonic fluctua-

tions. From $\lambda = 0$ to $\lambda = 1$, new-task success increases from 0.63% to 35.63%, whereas old-task success changes from 70.19% to 67.31%. Below 60% ER weight, acquisition is substantially reduced: even the equal-weight merge reaches only 24.38% new-task success, 11.25 percentage points below pure ER, while improving old-task success by 2.12 points.
Acquisition gains become smaller beyond approximately 60% ER weight, although the highest measured new-task success occurs at 100%. At 60%–70%, new-task success reaches 30.00%–30.63% and old-task success is already close to pure ER. At 80%, old-task success matches ER at 67.31%, while new-task success reaches 33.13%, only 2.50 points below ER. Thus, placing substantial weight on the pre-update checkpoint incurs a large acquisition cost, whereas mixtures dominated by the ER checkpoint approach its performance and offer little additional retention benefit. Across the tested coefficients, we observe no mixture that improves both old- and new-task success over directly using ER.

B.7 LoRA Rank and Learning-Rate Ablations

We examine whether increasing the adapter rank and adapter learning rate resolves the acquisition deficit of ER with LoRA on the RoboCasa365 Phase1-to-Phase2 transition. We compare language-model ranks $r = 16$ and $r = 128$, with action-expert ranks $2r$, and adapter learning-rate multipliers $0.2\times$, $1\times$, $5\times$, and $10\times$. These multipliers apply only to LoRA adapter parameters; all other trainable modules retain the unscaled reference schedule within this ablation (see Appendix A.3 for the trainable modules). Table 10 reports old- and new-task success separately. All configurations in this ablation, including the full-model ER reference, use batch size 128 and 10k optimizer steps, whereas the main experiments use batch size 512 and 15k steps per continual update. The ablation therefore processes one-sixth as many training examples (1.28 million versus 7.68 million, counting repeated sampling). These smaller-budget runs are not replications of the main results at the same rank and LR multiplier; their absolute success rates must be interpreted in light of this budget difference. The full-model ER reference below belongs to the high-rank ablation and uses its training budget.

Table 10: LoRA rank and learning-rate ablations with ER at batch size 128 and 10k steps. For LoRA, LR multipliers apply only to adapter parameters; other trainable modules retain the reference schedule. Success rates are percentages. Old and new tasks are evaluated separately.

Configuration	LR multiplier	Old tasks (65)	New tasks (20)
LoRA $r = 16$	$0.2\times$	66.63	11.88
	$1\times$	68.37	**18.44**
	$5\times$	67.50	15.63
	$10\times$	67.88	16.88
Full-model ER	$1\times$	63.08	**25.00**
LoRA $r = 128$	$0.2\times$	68.56	15.31
	$1\times$	69.90	19.06
	$5\times$	64.81	**21.88**
	$10\times$	12.12	0.63

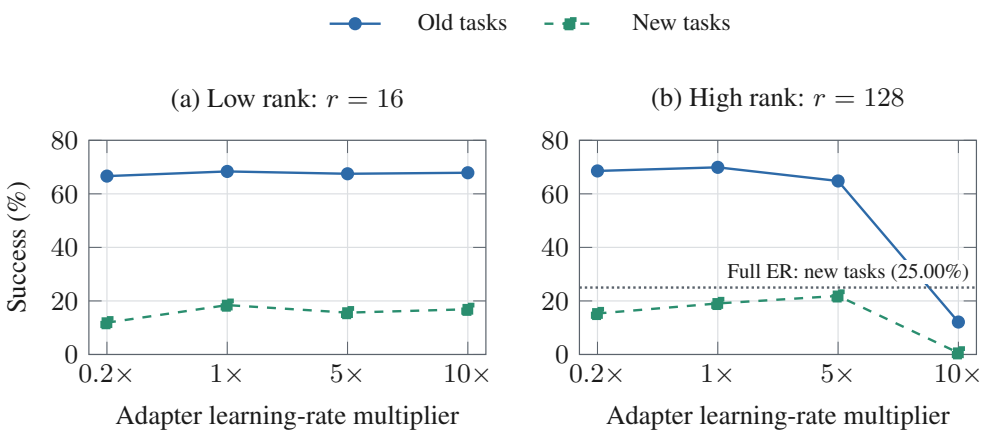


Figure 13: LoRA adapter learning-rate sensitivity (batch size 128, 10k steps). Circles and squares show old- and new-task success, respectively. The dotted line marks full-model ER in the high-rank ablation.

Low rank limits multi-task acquisition. The low-rank configuration, representative of the small adapters used in prior VLA CL studies (He et al., 2026; Chen et al., 2026), preserves old-task success at 66.63%–68.37% but reaches at most 18.44% on new tasks. Increasing the adapter learning rate to 5× or 10× does not improve acquisition over 1×. Thus, adapter learning-rate tuning alone does not resolve the weak acquisition of multiple new tasks at low rank within the tested range.

High rank benefits from a higher adapter learning rate, but still trails ER. At $r = 128$, raising the adapter learning rate from 1× to 5× improves new-task success from 19.06% to 21.88%. However, it remains 3.13 percentage points below full-model ER's 25.00%, while old-task success is only 1.73 points higher (64.81% versus 63.08%). At 10×, both old- and new-task performance collapse. High rank with 5× adapter LR therefore gives the strongest LoRA acquisition among the tested configurations, but does not close the gap to full-model ER.

Training-compute estimate. Fewer trainable parameters do not imply proportionally lower training compute. Frozen base projections still require a forward pass and input-gradient propagation; LoRA adds trainable low-rank projections. We estimate the dominant matrix-multiplication FLOPs using OpenPI's per-head attention adapters, feed-forward adapters, and block activation recomputation. This is an analytical estimate, not measured runtime or a compiled-kernel profile.
For a projection with input width d, output width k, n tokens, and adapter rank r, counting a multiply-add as two FLOPs gives

$$F_{\text{base}} = 2ndk, \qquad F_{\text{adapter}} = 2nr(d+k). \tag{12}$$

Without recomputation, training a dense projection costs approximately $3F_{\text{base}}$, whereas a frozen projection with LoRA costs $2F_{\text{base}} + 3F_{\text{adapter}}$. With one additional forward recomputation, these become $4F_{\text{base}}$ and $3F_{\text{base}} + 4F_{\text{adapter}}$, respectively. Attention adapters are counted separately for each head, matching OpenPI's factorization rather than treating the heads as one flattened projection. For the numerical estimate, we use the three-view image input of our training configuration: three 224×224 images (256 tokens each), together with 200 padded prompt tokens and the 32-action chunk used on RoboCasa. The language model has 18 layers with width 2048 and MLP width 16384; the action expert has 18 layers with width 1024 and MLP width 4096. Both use eight query heads, one key/value head, and head dimension 256. The trainable vision encoder is SigLIP So400m/14 (27 layers, width 1152, MLP width 4304). We include dense attention products, image and action projections, timestep projections, and adaptive-normalization projections. Optimizer operations, elementwise operations, communication, and evaluation are excluded. The compute comparison holds batch sizes and update counts equal across methods.
Let F_{B} denote the forward FLOPs of the frozen base projections inside recomputed blocks, F_{A} the adapter forward FLOPs, and F_{C} the unchanged forward FLOPs of attention products and the trainable vision blocks. Up to the small projections outside these blocks, the training-compute ratio is

$$\rho \simeq \frac{3F_{\text{B}} + 4F_{\text{A}} + 4F_{\text{C}}}{4F_{\text{B}} + 4F_{\text{C}}}. \tag{13}$$

At $r = 128$, these terms are approximately 3.857, 0.443, and 0.803 TFLOPs per example, respectively. Table 11 includes the small additional projections in the final totals.

Table 11: Estimated training FLOPs per example, including one full forward recomputation of each transformer block. These are arithmetic estimates under the stated input shapes, not timing measurements.

Configuration (language/action rank)	TFLOPs/example	Relative to full ER
Full-model ER	18.65	100.0%
LoRA 16/32 + ER	15.02	80.5%
LoRA 128/256 + ER	16.57	88.8%
LoRA 256/512 + ER (compute only)	18.34	98.3%

For the same three-view input, omitting block recomputation in both methods gives a high-rank estimate of 81.9%, illustrating the dependence of the arithmetic cost on the recomputation policy. Under the full-block recomputation estimate, high-rank LoRA saves only about 11% of training FLOPs, and doubling the ranks brings its cost close to full-model ER. This estimate does not quantify optimizer-state memory savings or wall-clock speed.
Taken together, high rank and a higher adapter learning rate improve LoRA's ability to acquire new tasks, but this stronger configuration still falls short of full-model ER in acquisition while approaching its training compute. The modest retention advantage in this single-update ablation accompanies

an acquisition deficit and limited arithmetic savings. It does not establish durable retention across successive updates (Section 5.2), supporting full-model ER as our preferred practical configuration for phase-wise continual learning.

B.8 LoRA Across Multiple Continual Updates

We compare full-model ER with LoRA at $1\times$ and $5\times$ adapter LR across the four-phase stream. Only adapter learning rates are scaled; other trainable modules retain the reference schedule. Both LoRA configurations use ranks 128/256 for the language model/action expert, start from full-SFT Phase1 weights, and merge adapters before initializing fresh adapters for each subsequent phase. These main-stream results use batch size 512 and 15k steps per continual update; the single-update ablations in Appendix B.7 instead use batch size 128 and 10k steps.

Table 12: Success rates (%) after each update, with 95% confidence-interval half-widths (pp), using the definition in Appendix A.4.1. All values are displayed to one decimal place. Phases are evaluated separately; LoRA labels denote adapter LR multipliers.

After	Method	P1 (65 tasks)	P2 (20 tasks)	P3 (20 tasks)	P4 (20 tasks)
Phase2	ER	67.3 ± 0.9	35.4 ± 1.8	–	–
	LoRA $1\times$ + ER	68.7 ± 0.9	17.1 ± 1.3	–	–
	LoRA $5\times$ + ER	68.8 ± 1.0	34.7 ± 1.8	–	–
Phase3	ER	68.0 ± 0.9	35.8 ± 1.7	18.2 ± 1.4	–
	LoRA $1\times$ + ER	68.8 ± 0.9	21.6 ± 1.5	4.3 ± 0.7	–
	LoRA $5\times$ + ER	66.8 ± 1.0	31.1 ± 1.8	14.7 ± 1.4	–
Phase4	ER	66.1 ± 0.9	35.3 ± 1.7	18.3 ± 1.4	7.9 ± 0.9
	LoRA $1\times$ + ER	67.8 ± 0.9	17.4 ± 1.4	4.1 ± 0.7	1.3 ± 0.4
	LoRA $5\times$ + ER	66.3 ± 1.0	27.7 ± 1.7	10.5 ± 1.2	4.1 ± 0.8

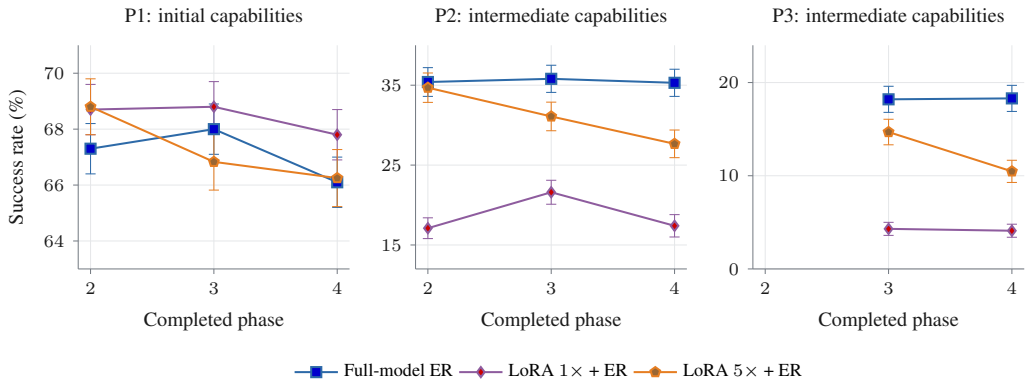


Figure 14: Per-phase success through subsequent updates. Each curve starts after the phase has been learned; error bars are 95% confidence intervals. Separate axes preserve the distinction between task groups. LoRA labels denote adapter LR multipliers.

Improved acquisition does not ensure retention across phases. At $5\times$ adapter LR, LoRA initially reaches 34.7% on Phase2, close to full-model ER's 35.4%, while retaining 68.8% on Phase1 versus 67.3% for ER. Its acquisition of Phases 3 and 4 remains lower (14.7% and 4.1%, versus 18.2% and 7.9%). These results substantially improve on LoRA at $1\times$ but do not close the later-phase acquisition gap.

The higher adapter learning rate also exposes a retention limitation. Phase2 success falls from 34.7% after acquisition to 31.1% after Phase3 and 27.7% after Phase4, a cumulative loss of 7.0 pp. Phase3 success falls from 14.7% to 10.5%, a loss of 4.2 pp. Full-model ER remains nearly unchanged on these intermediate phases (35.4% to 35.3% for P2 and 18.2% to 18.3% for P3). Final Phase1 success remains similar (66.3% versus 66.1%), so the retention deficit concerns the capabilities acquired in intermediate phases rather than uniformly worse retention of all old tasks.

Using the common Phase1 reference of 67.1% and each method's own acquisition scores, task-averaged BWT for LoRA $5\times$ is approximately $+1.7$, -1.1, and -2.7 pp after Phases 2, 3, and 4. BWT is computed from the original reported rates before rounding values for display. By contrast, LoRA $1\times$ has favorable BWT but much weaker acquisition (Appendix B.1). The pattern motivates distinguishing preservation of initial full-SFT capabilities from retention of capabilities

learned through successive LoRA updates; interference between those updates remains a possible explanation rather than an established mechanism.

B.9 New-Task Acquisition in RoboCasa365 Task Mixtures

We report the full new-task acquisition and old-task retention results from the RoboCasa365 task-mixture experiment in Section 4.3 and Figure 4(a). T, K, P, and C denote `TurnOffSimmeredSauceHeat`, `KettleBoiling`, `PrepareCoffee`, and `ClearCuttingBoard`, respectively. Table 13 reports each trained new task over 128 evaluation episodes and averages success equally over the new tasks in that row. Old-task retention averages the 65 Phase1 tasks, with 16 episodes per task (1,040 in total). The experiment uses 2k-step SFT without replay. New and old tasks are reported separately.

Table 13: RoboCasa365 task mixtures: new-task acquisition and old-task retention. All entries are success rates (%); dashes denote tasks outside that row's training set, not zero success.

Training tasks	T	K	P	C	New-task mean	Old-task retention
T	10.94	–	–	–	10.94	4.52
K	–	22.66	–	–	22.66	25.00
C	–	–	–	49.22	49.22	25.38
T + K	8.59	17.19	–	–	12.89	23.37
T + P	13.28	–	7.03	–	10.16	15.10
T + C	13.28	–	–	28.91	21.09	30.87
P + C	–	–	7.81	46.09	26.95	31.15
T + K + P	9.38	22.66	3.91	–	11.98	23.75
T + P + C	10.16	–	3.91	21.88	11.98	9.62
T + K + P + C	10.94	12.50	1.56	31.25	14.06	23.17

Retention is not ordered by weaker acquisition. We examine whether higher retention simply reflects poorer acquisition of the new tasks. The results do not show a consistent inverse relationship of this kind. At the same task count, T+P and T+C achieve identical success on their shared task T (13.28%), yet old-task retention is 15.10% and 30.87%, respectively. T+C also has higher mean new-task success (21.09% versus 10.16%). Thus, its higher retention does not coincide with lower measured acquisition, including on the directly comparable shared task.

Adding P to T+C provides a complementary comparison. Success on both shared new tasks decreases: T falls from 13.28% to 10.16%, and C from 28.91% to 21.88%. Old-task retention also falls, from 30.87% to 9.62%. Here, weaker acquisition accompanies worse retention rather than preserving old capabilities. Moreover, T+K+P and T+P+C have the same mean new-task success (11.98%), but retain 23.75% and 9.62% on old tasks, a 14.13-point difference. These observations support the main text's conclusion that the retention outcome depends on the training mixture. These results argue against weaker acquisition as the sole explanation for better retention.