HIERARCHICAL REPLAY MEMORY CONSTRUCTION FOR CONTINUAL IMITATION LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Experience replay (ER) offers a simple and effective strategy for continual imitation learning, but its effectiveness depends critically on what is retained under a limited budget. Robot demonstrations provide complementary supervision across granularities: complete trajectories preserve execution context across task stages, recurring behaviors and transitions form shared structure across demonstrations, and local windows capture fine-grained state–action variation within these behaviors. Although prior work has explored data selection at different temporal scales, how to jointly organize them within replay memory remains underexplored. To address this gap, we propose Hierarchical Memory Retention (HMR), a hierarchical replay memory construction framework that progressively organizes retention from global context to local variation under a fixed budget. HMR first constructs a task-progress-aware behavioral structure by aligning demonstrations along task progress and clustering local action–state sequences, revealing recurring modes, stage-specific occurrences, and transitions. Guided by this structure, HMR retains a small set of complete demonstrations as contextual anchors and supplements them with local windows for reliable occurrences and transitions they underrepresent, establishing task-wide structural coverage. Remaining capacity is allocated according to behavioral frequency, event importance, and current coverage to preserve representative local variations. Extensive experiments on LIBERO task suites, together with cross-architecture and cross-domain evaluations, demonstrate consistent improvements over baseline methods under matched replay budgets, while ablations validate its key design choices.

1 INTRODUCTION

Robots require continual learning to adapt to evolving objects, tasks, and environments while retaining previously acquired skills. However, sequentially fine-tuning a single policy can overwrite previously learned behavioral representations, leading to catastrophic forgetting. Experience replay (ER) provides a simple yet effective approach to mitigate this problem by replaying a subset of past experiences alongside current data (Rolnick et al., 2019; Zhou & Cao, 2021; Wan et al., 2024). Under a limited memory budget, its effectiveness depends on how replay memory is constructed.

Robot demonstrations encode state–action supervision with temporal dependencies across task stages (Liu et al., 2023). As shown in Figure 1(a), such knowledge is embedded in temporally structured and highly nonuniform demonstrations. At the trajectory level, complete demonstrations preserve task-level context by retaining the progression of behaviors and intermediate states. Across demonstrations, shared behavioral modes and transitions provide a basis for identifying redundant supervision and gaps in retained structural support. Within these shared structures, however, local state–action realizations can still vary, motivating complementary windows that preserve representative within-structure variation. Effective replay memory should therefore jointly preserve task-level context, shared behavioral structure, and representative local variation.

Recent robot-data curation methods have explored data value at different temporal and structural scales. Demonstration-level approaches evaluate complete trajectories using state–action informativeness or their contribution to downstream policy performance (Hejna et al., 2025; Agia et al., 2025; Dass et al., 2026), while finer-grained methods identify suboptimal or redundant state–action pairs and informative frames (Zhang et al., 2025b; Yu et al., 2026). More recent work has introduced

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

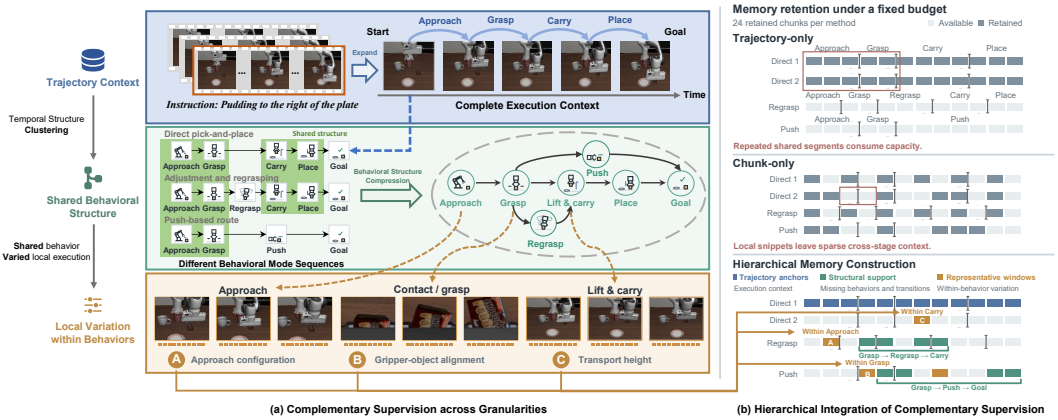

Figure 1: **Motivation for hierarchical replay-memory construction. (a)** Demonstrations provide complementary supervision through execution context, shared structure across alternative behavioral sequences, and within-behavior variation. **(b)** Under an equal replay budget, HMR combines trajectory anchors, structural support, and representative windows, addressing redundant segments in trajectory-only retention and fragmented context in chunk-only retention.

intermediate temporal and behavioral structure to guide trajectory selection (Wu et al., 2026) and phase-level replay allocation (Chen et al., 2026b), enabling better structural coverage.

Despite these advances, data retention is still typically organized around a dominant granularity. Preserving execution context requires retaining how behaviors connect across stages, whereas demonstrations following similar behavioral sequences may differ mainly in short segments. Retaining whole trajectories to capture these local differences can therefore duplicate well-represented segments, whereas retaining only local windows may underrepresent execution context and cross-stage behavioral connections. How to jointly exploit the distinct supervision available across granularities to preserve a more complete task structure under a limited replay budget remains underexplored.

Based on these observations, we propose **Hierarchical Memory Retention (HMR)**, a replay memory construction framework that progressively preserves task-level context, recurring behavioral structure, and representative local variation, as shown in Figure 1(b). By aligning task progress across demonstrations and grouping similar local state–action sequences, HMR identifies recurring behaviors and their stage-specific occurrences within each task, providing a common reference for assessing what the selected memory already represents. This structure guides the selection of a small set of complete demonstrations as contextual anchors, providing broad structural support and preserving supervision along complete executions. A bounded coverage stage then supplements these anchors with local windows covering occurrences and transitions supported across demonstrations but underrepresented by the anchors, forming a compact structural scaffold.

To capture local state–action variation within these structures, HMR allocates the remaining capacity across the resulting behavioral groups according to their prevalence, event importance, and the support already retained. Within each group, HMR selects representative windows that provide complementary variation relative to the current task's selected set, while limiting concentration within individual demonstrations and among temporally adjacent windows. Together, these stages retain complementary supervision under a fixed per-task memory budget.

In summary, our primary contributions are:

- We introduce a hierarchical perspective on replay-memory construction for continual imitation learning, emphasizing the coordinated retention of execution context, behavioral structure, and within-structure variation under a fixed per-task budget.
- We propose HMR, which discovers temporal behavioral structure through task-progress alignment and clustering to guide memory construction from global context to local variation. It combines complete-demonstration anchors, targeted structural coverage, and representative completion, with budget allocation and window selection conditioned on retained support.
- Experiments on LIBERO and RoboTwin-CL demonstrate improved continual-learning performance across policy architectures under matched replay budgets, with ablations supporting the key design choices.

2 Related Work

Continual imitation learning. Continual imitation learning aims to acquire new manipulation skills from sequential demonstrations while retaining previously learned behaviors (Liu et al., 2023). Existing approaches mitigate catastrophic forgetting by constraining changes to previously acquired knowledge (Kirkpatrick et al., 2017; Roy et al., 2025) or by isolating or expanding model capacity across tasks (Mallya & Lazebnik, 2018; Liu et al., 2024; Lee et al., 2024; Zhang et al., 2025a; Xu & Nie, 2025; Römer et al., 2026). Experience replay (ER) preserves historical supervision by revisiting stored experiences, with subsequent work improving memory selection or retrieval through diversity and interference-aware criteria (Rolnick et al., 2019; Aljundi et al., 2019b;a). In continual robot learning, PHASER (Chen et al., 2026b) further organizes replay at the manipulation-phase level and adapts replay support according to interference risk. We instead construct replay memory hierarchically, coordinating supervision across temporal granularities.

Data selection for robot learning. Robot data selection has been explored at multiple levels of temporal granularities. Demonstration-level methods evaluate entire trajectories based on data informativeness or downstream policy contribution (Hejna et al., 2025; Dass et al., 2026; Agia et al., 2025), while finer-grained approaches curate individual state–action transitions or informative frames to reduce redundancy and preserve critical interactions (Zhang et al., 2025b; Yu et al., 2026). SIEVE (Wu et al., 2026) further models reusable visuomotor primitives and their transitions, showing that long-horizon behavioral structure provides additional information beyond isolated samples or whole-trajectory scores. Building on these insights, HMR jointly preserves execution context, behavioral structure, and representative local variation under a fixed per-task replay budget.

Temporal structure in robot demonstrations. Robot demonstrations exhibit rich temporal structure that can be decomposed into meaningful behavioral units. Early approaches identify subtask boundaries through recurring transition states or weakly supervised temporal alignment (Krishnan et al., 2017; Shiarlis et al., 2018), while latent-variable and skill-discovery methods further infer reusable behaviors from unsegmented demonstrations for hierarchical composition and control (Kipf et al., 2019; Shankar & Gupta, 2020; Zhu et al., 2022; Wan et al., 2024). Recent approaches further organize demonstrations using cross-embodiment skill representations, semantic guidance, or annotated skill decompositions (Xu et al., 2023; Kou et al., 2024; Garrett et al., 2024; Wan et al., 2025). Together, these studies highlight the importance of temporal boundaries, recurring behaviors, and their structural organization in robot demonstrations. Unlike methods for skill discovery or hierarchical control, HMR uses temporal structure to guide replay-memory construction.

3 Hierarchical Replay Memory Construction

3.1 Problem Formulation

We consider continual imitation learning over a sequence of manipulation tasks. For task k, let C^k denote the training chunks extracted from N expert demonstrations. Each chunk $c \in C^k$ contains H consecutive observation–action pairs and retains its source-task instruction. With experience replay, the policy learns from current-task data and historical chunks using its native training objective. After training on task k, a subset $S^k \subseteq C^k$ with $|S^k| = b$ is retained, where b is a fixed per-task quota measured in chunks, counting overlapping chunks separately. The selected chunks are appended to replay memory for subsequent tasks, while previously retained chunks remain unchanged. We construct S^k that to preserve useful behavioral supervision for subsequent learning under this quota.

To this end, HMR first discovers recurring behavioral modes and their temporal organization across demonstrations of the current task. The resulting temporal behavioral structure guides hierarchical memory construction from global execution context through structural coverage to representative local variation, with finer-grained allocation and selection conditioned on already retained support.

3.2 Temporal Behavioral Structure Discovery

Demonstrations of the same task contain recurring local patterns, termed behavioral modes. The same mode may recur at different stages with distinct observations and preceding or subsequent

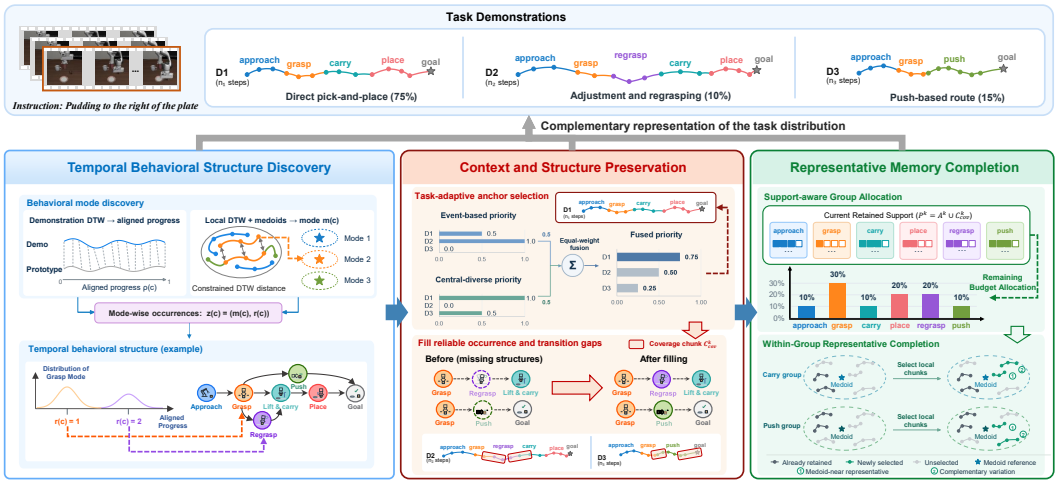

Figure 2: **Overview of HMR.** *Left:* Temporal behavioral structure discovery. Task-progress alignment and local sequence clustering identify recurring modes and stage-specific occurrences, linked by observed transitions to provide a structural reference for memory selection. *Middle:* Context and structure preservation. Guided by this structure, complete-demonstration anchors preserve coherent execution context, while targeted chunks supplement reliable occurrences and transitions underrepresented by the anchors. *Right:* Representative memory completion. Conditioned on retained support, group-wise allocation and within-group selection use the remaining budget to capture representative local variation that complements the supervision already preserved.

behaviors, so mode-level coverage alone does not imply adequate support for each stage-specific occurrence. We therefore assign each training chunk c a joint structural label

$$z(c) = \big(m(c), r(c)\big), \tag{1}$$

where $m(c)$ denotes its behavioral mode and $r(c)$ identifies the stage-specific occurrence of that mode. Within each task, chunks sharing a label form a behavioral group z across demonstrations. These groups are the nodes of a directed graph, with edges $z \to z'$ recording observed transitions between distinct groups along a demonstration. This graph defines our temporal behavioral structure for assessing retained support for individual occurrences and transitions.

Discovering behavioral modes and temporal occurrences. Corresponding stages may occur at different normalized times across demonstrations. We therefore use dynamic time warping (DTW) to establish a common task-progress reference.

Each demonstration is represented as a sequence of descriptors comprising actions, gripper and joint states, and their first-order temporal differences. We initialize a task-specific feature prototype by averaging uniformly resampled descriptor sequences, then alternate DTW alignment with prototype updates that average the matched features across demonstrations. Each chunk inherits an aligned task-progress coordinate $\rho(c)$ from the final DTW alignment of its starting frame to this prototype.

Using $\rho(c)$, we construct a discovery pool stratified by demonstration and progress interval. Separately, within this pool, we compare local state–action sequences using constrained DTW. The behavioral distance $D_{\mathrm{DTW}}(c, c')$ is the accumulated cost of the minimum-cost alignment divided by its path length. Task progress is excluded from this local distance.

We apply medoid-based clustering with different candidate mode counts. For each candidate clustering, we smooth the aligned-progress distribution within each mode and identify separated peaks with sufficient cross-demonstration support as occurrence centers. A mode retains a single occurrence when the evidence does not support multiple occurrences. Candidate mode counts are evaluated using the resulting occurrence structure and its cross-demonstration support. Let $\bar{c}_m$ denote the selected medoid of mode m and $\xi_{m,r}$ its r-th occurrence center. Every task chunk is assigned labels

$$m(c) = \arg\min_{m} D_{\mathrm{DTW}}(c, \bar{c}_m), \qquad r(c) = \arg\min_{r} \big|\rho(c) - \xi_{m(c),r}\big|. \tag{2}$$

Extracting transitions and assessing structural reliability. Coverage of individual occurrences does not establish whether their temporal connections are retained. We therefore traverse chunks in the original temporal order of each demonstration and record a directed transition $z \to z'$ whenever consecutive joint labels differ. The starting frame of the chunk carrying the new label defines the transition boundary. A chunk from the same demonstration supports this transition when its temporal span contains the boundary, linking graph edges to candidate replay chunks.

We define U_{rel} as the occurrence and directed-transition types supported by at least $n_{\min}$ distinct demonstrations, counting each demonstration once and computing support over all current-task candidates. This prevents overlapping chunks from inflating structural reliability.

Separately, each chunk receives a dense importance score combining local behavioral changes, boundary cues in the feature sequences, and agreement across aligned demonstrations. Cross-demonstration support determines eligibility as a structural coverage target, while importance provides a complementary priority signal for subsequent memory selection.

3.3 Context and Structure Preservation

Based on the temporal behavioral structure identified above, we first select a small set of complete demonstrations as anchors to preserve supervision throughout the task execution. Since these anchors may not cover all reliable structures, we then add a bounded set of chunks to fill the remaining coverage gaps. Let A^k denote all training chunks from the selected anchor demonstrations and C^k_{cov} the additional chunks selected for structural coverage. Their union $P^k = A^k \cup C^k_{\text{cov}}$ is protected during subsequent memory completion. Memory usage is counted in chunks, including those from complete demonstrations, while each policy training sample remains a length-H chunk.

Task-adaptive selection of demonstration anchors. We construct candidate anchor sets from two complementary rankings. The event-based ranking favors demonstrations that capture recurring behavioral changes, whereas the central-diverse ranking favors representative trajectories with broad coverage of the demonstration set. Since their relative utility can vary across tasks, both anchor sets are evaluated against the same reference structure, initialized from an equal-weight combination of the two rankings. We compare their coverage of reliable occurrences and transitions, together with the additional memory budget required to recover missing support. One ranking is preferred only if it is no worse on every criterion and strictly better on at least one.

To assess the robustness of this preference to demonstration subsampling, we repeat the comparison over multiple subsampling trials. We adopt a ranking only when its preference meets the stability threshold across a sufficient number of completed trials; otherwise, we retain the equal-weight combination. Demonstrations are then selected in ranked order to meet the prescribed anchor budget.

Saturating structural coverage. The coverage stage targets structural gaps left by the anchor demonstrations. Let V^k denote the coverage requirements induced by U_{rel}, with separate chunk-count and distinct-demonstration requirements for each reliable structural element. For each $v \in V^k$, let $n_v(S)$ and κ_v denote its support within a selected chunk set S and its target support, respectively. Depending on the requirement, $n_v(S)$ counts either supporting chunks or their distinct source demonstrations. We define the capped coverage objective as

$$F_{\text{cov}}(S) = \sum_{v \in V^k} \min\{\kappa_v, n_v(S)\}. \tag{3}$$

With the structural labels and support targets fixed, $F_{\text{cov}}(S)$ is monotone and submodular (Krause & Golovin, 2014). Each term saturates at its target, so further support for that requirement contributes no additional gain.

Starting from $S = A^k$, we greedily add the eligible chunk that yields the largest positive increase in $F_{\text{cov}}(S)$ at each step. Support counts include anchor chunks, with each demonstration counted at most once per distinct-demonstration requirement. A temporal-separation criterion discourages adjacent selections within each demonstration and is relaxed when necessary. Selection stops when the coverage budget is reached or no candidate provides positive marginal gain. These targets guide structural preservation rather than guarantee sufficient supervision for task execution.

3.4 Representative Memory Completion

With task context and reliable structural support already preserved, the remaining memory capacity is used to capture representative variation within each behavioral group, accounting for the windows already retained in P^k. The remaining budget is $b^k_{\text{rem}} = b - |P^k|$. Memory completion proceeds in two steps: we first determine how the remaining capacity is allocated across behavioral groups, and then select the specific chunks within each group.

Allocation by diminishing marginal utility. For each group g, let N_g denote the number of chunks in the current task, I_g its normalized importance, and p_g the number already retained in P^k. After allocating a_g additional slots to the group, the marginal utility of the next slot is

$$u_g = \frac{N_g(1+\eta I_g)}{\alpha + p_g + a_g}, \tag{4}$$

where $\eta \geq 0$ controls the contribution of importance and $\alpha > 0$ provides smoothing. Frequent and important groups receive higher priority, while groups already well represented in P^k receive lower marginal utility. As additional slots are assigned, their utility further decreases, encouraging the remaining capacity to spread across underrepresented groups.

Starting from $a_g = 0$, we repeatedly assign the next slot to the eligible group with the largest u_g, subject to the number of remaining candidate chunks in that group. This marginal allocation rule exactly maximizes the corresponding separable discrete concave utility under the total budget and group capacity constraints (Federgruen & Groenevelt, 1986). The result applies to the allocation subproblem with fixed groups and weights, rather than to the downstream policy objective.

Selection of representative and complementary chunks. Given the group-wise allocations, we initialize $S = P^k$. For each group with a positive quota, we first add the eligible chunk within that group nearest to the medoid of the corresponding behavioral mode, counting it toward the group's quota. We then iteratively add the highest-scoring eligible chunk from groups with remaining quota:

$$q(c \mid S) = w_{\text{cen}} s_{\text{cen}}(c) + w_{\text{evt}} s_{\text{evt}}(c) + w_{\text{div}} d(c,S) + w_{\text{demo}} g(c,S) - w_{\text{time}} o(c,S). \tag{5}$$

Here, $s_{\text{cen}}(c)$ and $s_{\text{evt}}(c)$ favor medoid proximity and event importance, respectively. The selection-dependent terms reward descriptor diversity through $d(c,S)$, favor less-represented demonstrations through $g(c,S)$, and penalize within-demonstration temporal proximity through $o(c,S)$. In particular, $g(c,S)$ decreases with the number of chunks already retained from the candidate's source demonstration. These terms are evaluated against the evolving set S, which includes anchor, coverage, and newly added chunks. Selection continues until all group-wise quotas are filled, yielding the final task memory $S^k = S$ with $|S^k| = b$.

4 Evaluation of Hierarchical Replay Memory Construction

4.1 Experimental Setup

Benchmarks and policies. We evaluate on LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long (Liu et al., 2023), each comprising an independent ten-task stream with 50 demonstrations per task and a fixed task order. Our main experiments use BCTransformer. To assess generality, we additionally evaluate StarVLA's Qwen-GR00T on LIBERO and Qwen-OFT on RoboTwin-CL (Community, 2026), an eight-task stream on the dual-arm RoboTwin 2.0 platform (Chen et al., 2026a).

Baselines and comparison protocol. With BCTransformer, we compare HMR against (i) **ER**, using deterministic prefix selection and uniform replay; (ii) **GSS** (Aljundi et al., 2019b), selecting chunks for gradient diversity; and (iii) **DemInf** (Hejna et al., 2025), ranking trajectories by state–action mutual information. The VLA experiments compare against ER. All comparisons use $b =$ 1,000 chunks per task, with matched policy-training and replay budgets within each comparison.

Evaluation metrics. We report forward transfer (**FWT**) for new-task learning, negative backward transfer (**NBT**) for average forgetting over subsequent stages, and area under the success-rate curve

Table 1: **Knowledge transfer and retention on LIBERO.** All methods use BCTransformer; bold denotes the best mean for each suite and metric. HMR achieves the highest mean FWT and AUC across all four suites.

	LIBERO-Spatial			LIBERO-Object		
Method	FWT ↑	NBT ↓	AUC ↑	FWT ↑	NBT ↓	AUC ↑
ER	64.06 ± 2.59	23.72 ± 3.03	58.53 ± 0.46	74.32 ± 1.59	21.78 ± 4.93	68.21 ± 4.15
GSS	68.17 ± 2.39	20.52 ± 2.93	65.04 ± 4.32	75.86 ± 2.76	13.87 ± 4.34	76.04 ± 5.89
DemInf	62.99 ± 4.96	25.37 ± 3.14	56.40 ± 5.89	73.80 ± 1.62	20.47 ± 2.42	69.50 ± 3.65
HMR	$\mathbf{69.03 \pm 2.84}$	$\mathbf{18.58 \pm 2.28}$	$\mathbf{65.99 \pm 2.05}$	$\mathbf{77.17 \pm 2.50}$	$\mathbf{11.67 \pm 2.47}$	$\mathbf{80.81 \pm 4.31}$
	LIBERO-Goal			LIBERO-Long		
Method	FWT ↑	NBT ↓	AUC ↑	FWT ↑	NBT ↓	AUC ↑
ER	60.52 ± 2.90	29.68 ± 6.41	49.40 ± 5.59	37.96 ± 6.84	23.45 ± 4.02	29.50 ± 5.17
GSS	64.18 ± 3.38	24.21 ± 2.35	59.26 ± 5.13	46.65 ± 1.95	$\mathbf{19.97 \pm 1.42}$	43.04 ± 0.78
DemInf	62.70 ± 4.13	29.29 ± 1.62	52.13 ± 4.56	37.64 ± 2.32	25.01 ± 4.28	29.93 ± 0.43
HMR	$\mathbf{67.77 \pm 4.05}$	$\mathbf{19.74 \pm 4.43}$	$\mathbf{64.42 \pm 4.40}$	$\mathbf{51.50 \pm 3.66}$	24.31 ± 4.10	$\mathbf{46.23 \pm 1.96}$

(**AUC**) for overall acquisition and retention. VLA experiments report final-policy average success rate (**ASR**) instead of FWT. BCTransformer AUC includes within-task learning, whereas VLA AUC uses stage-end evaluations following CLARE (Römer et al., 2026). Higher FWT, ASR, and AUC and lower NBT are better. Results are means $\pm$ sample standard deviations over three seeds for the main BCTransformer comparison and two for VLA and ablation experiments. Full configurations, baseline adaptations, and metric definitions are provided in Appendix B.

4.2 Knowledge Transfer and Retention on LIBERO

Table 1 shows that HMR achieves the highest mean FWT and AUC across all four LIBERO suites. On average across suites, HMR improves FWT and AUC over ER by 7.15 and 12.95 points, respectively, while reducing NBT by 6.08 points. Compared with GSS, the strongest baseline in FWT and AUC, HMR further improves these metrics by 2.65 and 3.52 points. These results demonstrate improved acquisition and overall continual-learning performance under the same replay budget.

The baseline comparison in Table 1 shows that chunk-level selection performs better than trajectory-level selection across all four suites. This supports our motivation that retaining entire trajectories can spend substantial capacity on repeated behavioral segments, limiting the amount of complementary supervision preserved within a fixed budget. Chunk-level selection offers greater flexibility to retain informative local segments, but does not explicitly preserve the execution context and behavioral connections between them. HMR combines complete-demonstration anchors with coverage support for missing occurrences and transitions, then retains representative local variations. Its further improvements over chunk-level selection support the value of this coordinated construction: preserving task context, structural connections, and local detail together produces a more effective replay memory than relying on either granularity alone.

4.3 Generality across Policy Architectures and Task Domains

Cross-Architecture generality. With Qwen-GR00T, HMR achieves higher mean ASR and AUC and lower mean NBT than ER across all four LIBERO suites (Figure 3(a)). Average ASR and AUC gains are 5.68 and 4.29 points, respectively; on LIBERO-Long, ASR improves by 9.10 points and NBT decreases from 19.15 to 8.34. These gains demonstrate that the benefits of hierarchical replay retention extend beyond BCTransformer to policies with pretrained representations and flow-matching action generation.

Cross-domain generality. RoboTwin-CL extends evaluation to single-arm and coordinated bimanual manipulation with scene-dependent arm selection, using the regression-based Qwen-OFT policy. Figure 3(b) shows that HMR improves mean ASR and AUC over ER by 5.50 and 2.84

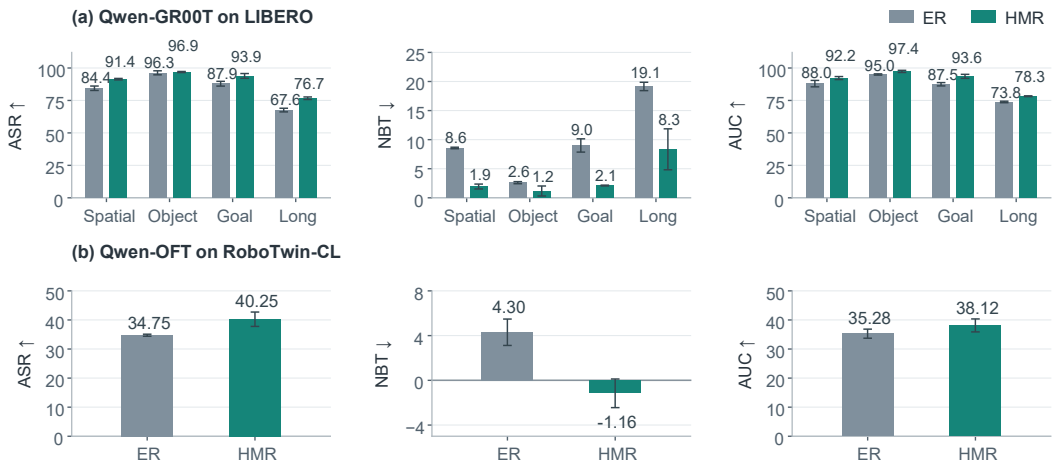

Figure 3: **Generality across policy architectures and task domains.** We compare HMR with ER using (a) Qwen-GR00T on LIBERO and (b) Qwen-OFT on RoboTwin-CL. Across all evaluated task streams, HMR achieves higher mean ASR and AUC and lower mean NBT.

points, respectively, while reducing NBT from 4.30 to −1.16, indicating positive backward transfer on average. The gains under these heterogeneous coordination requirements, together with the LIBERO results, demonstrate that hierarchical retention remains effective across distinct policy architectures and manipulation domains.

4.4 Analysis of Memory Construction Design Choices

We examine the contributions of the three retention stages using BCTransformer on LIBERO-Goal and LIBERO-Spatial (Goal and Spatial). All variants share the same replay quota; component comparisons use A25 on Goal and A15 on Spatial.

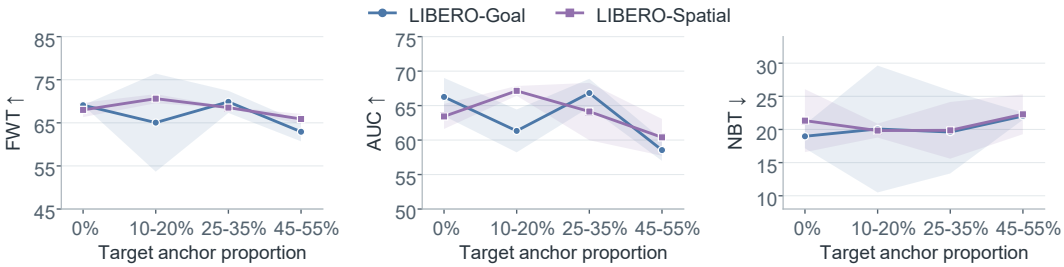

Figure 4: **Sensitivity to the target anchor proportion.** The coverage and completion procedures remain unchanged. The highest mean FWT and AUC occur at different target proportions on Goal and Spatial.

Sensitivity to anchor proportion. With coverage and completion retained, the highest mean FWT and AUC occur at anchor targets of 25–35% (A25) on Goal and 10–20% (A15) on Spatial (Figure 4). These configurations improve AUC over no anchors by 0.56 and 3.71 points, respectively, whereas increasing the target to 45–55% lowers AUC by 8.27 and 6.74 points relative to these configurations. The results favor preserving complete execution context through a limited anchor allocation while reserving capacity for structural and local supervision. Notably, the highest mean AUC occurs at different anchor targets on Goal and Spatial, suggesting a task-dependent balance between contextual anchors and complementary local chunks.

Effect of structural coverage. We remove explicit coverage and reassign its capacity to representative completion, keeping the anchors and discovered structure unchanged. Restoring coverage improves mean FWT by 3.57 and 3.81 points and AUC by 3.39 and 1.19 points on Goal and Spatial, respectively (Figure 5). Under the same replay budget, adding explicit coverage yields higher FWT

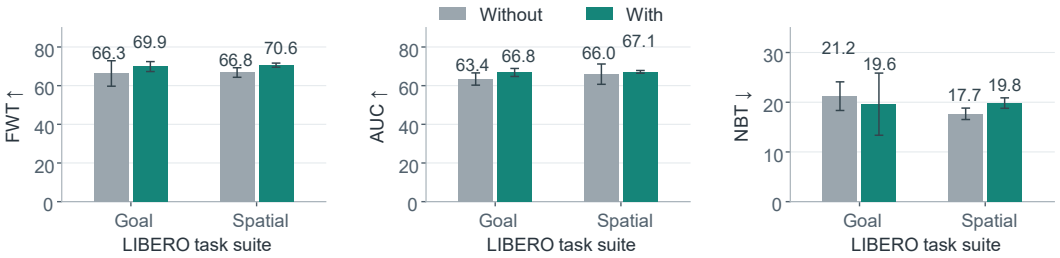

Figure 5: **Effect of structural coverage.** Anchors remain fixed, and removing coverage reallocates its capacity to representative completion. Explicit coverage improves mean FWT and AUC on both suites.

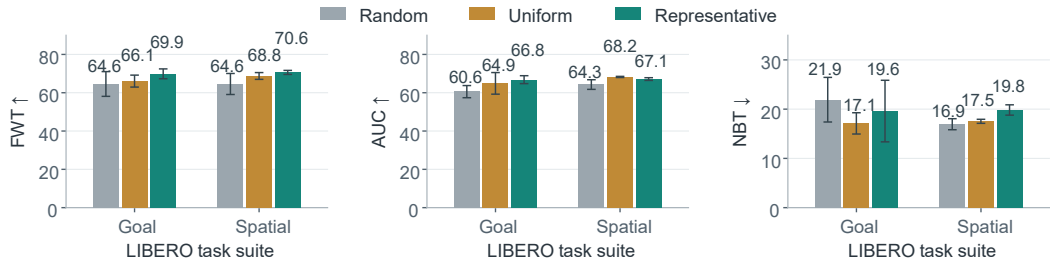

Figure 6: **Effect of representative memory completion.** Random, Uniform, and representative completion share the same anchors and coverage chunks. Representative completion achieves the highest mean FWT on both suites, while its AUC ranking relative to Uniform is task-dependent.

and AUC than anchor retention and representative completion alone. These gains highlight the value of retaining reliable occurrences and transitions that remain underrepresented by the anchors.

Effect of representative completion. With anchor and coverage chunks fixed, representative completion achieves higher mean FWT and AUC than Random on both suites, with AUC gains of 6.25 and 2.87 points on Goal and Spatial, respectively (Figure 6). These gains highlight the value of using behavioral structure and retained support to guide selection of the remaining chunks. Compared with Uniform, FWT still increases by 3.80 and 1.86 points on Goal and Spatial, respectively, while AUC improves by 1.93 points on Goal but decreases by 1.10 points on Spatial. Uniform also yields lower mean NBT on both suites. One possible explanation for the differing AUC outcomes across suites is that structure-aware selection better supports goal-specific interactions, while uniform sampling remains competitive for Spatial's recurring pick-and-place tasks. Taken together, these results indicate that representative completion improves new-task acquisition over Random and Uniform, while its overall acquisition–retention benefit remains task-dependent.

5 Conclusion

We presented HMR, a hierarchical replay-memory construction framework for continual imitation learning under a fixed per-task budget. Guided by temporal behavioral structure discovered across demonstrations, HMR combines complete-demonstration anchors, explicit structural coverage, and representative completion, with finer-grained allocation and selection conditioned on retained support. Experiments on LIBERO and RoboTwin-CL demonstrate improved continual-learning performance across BCTransformer and two VLA policies with pretrained vision–language backbones. Together with the ablations, these findings substantiate our central premise that jointly preserving execution context, behavioral structure, and local variation enables more effective use of replay capacity. The task-dependent preferences revealed by our analysis further highlight opportunities to tailor memory construction to individual task requirements. Building on these findings, future work will investigate task-adaptive allocation and selection, moving beyond prescribed anchor targets to further improve the acquisition–retention balance.

AI USE STATEMENT

In this work, we used generative AI tools for manuscript editing and routine code refactoring, including improving language clarity and reorganizing existing code. All AI-assisted text was reviewed and revised by the authors, and code changes were manually inspected and tested for correctness. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To support reproducibility, we describe HMR's memory construction procedure in Sections 3.2-3.4 and provide implementation details and supporting mathematical analyses in Appendix A. Appendix B documents dataset preparation and task sequences, policy configurations, baseline adaptations, training and replay protocols, and hyperparameters for memory construction and ablations. It also specifies evaluation procedures, metric definitions, and the random seeds and aggregation rules used to report means and standard deviations. Appendix C provides the numerical results underlying the generality and ablation figures, together with task-wise acquisition and retention curves, to facilitate verification of the reported findings.

REFERENCES

Christopher Agia, Rohan Sinha, Jingyun Yang, Rika Antonova, Marco Pavone, Haruki Nishimura, Masha Itkina, and Jeannette Bohg. CUPID: Curating data your robot loves with influence functions. In *9th Annual Conference on Robot Learning*, 2025.

Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. Online continual learning with maximal interfered retrieval. *Advances in neural information processing systems*, 32, 2019a.

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32, 2019b.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. In *Forty-third International Conference on Machine Learning*, 2026a.

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

StarVLA Community. Starvla: A lego-like codebase for vision-language-action model developing. *arXiv preprint arXiv:2604.05014*, 2026.

Shivin Dass, Alaa Khaddaj, Logan Engstrom, Aleksander Madry, Andrew Ilyas, and Roberto Martín-Martín. Datamil: Selecting data for robot imitation learning with datamodels. In *International Conference on Learning Representations*, volume 2026, pp. 1311–1336, 2026.

Awi Federgruen and Henri Groenevelt. The greedy procedure for resource allocation problems: Necessary and sufficient conditions for optimality. *Operations research*, 34(6):909–918, 1986.

Caelan Reed Garrett, Ajay Mandlekar, Bowen Wen, and Dieter Fox. Skillmimicgen: Automated demonstration generation for efficient skill learning and deployment. In *8th Annual Conference on Robot Learning*, 2024.

Joey Hejna, Suvir Mirchandani, Ashwin Balakrishna, Annie Xie, Ayzaan Wahid, Jonathan Tompson, Pannag Sanketi, Dhruv Shah, Coline Devin, and Dorsa Sadigh. Robot data curation with mutual information estimators. *arXiv preprint arXiv:2502.08623*, 2025.

Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. Compile: Compositional imitation learning and execution. In *International Conference on Machine Learning*, pp. 3418–3428. PMLR, 2019.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Longxin Kou, Fei Ni, Yan Zheng, Jinyi Liu, Yifu Yuan, Zibin Dong, and Jianye Hao. Kisa: A unified keyframe identifier and skill annotator for long-horizon robotics demonstrations. In *Forty-first International Conference on Machine Learning*, 2024.

Andreas Krause and Daniel Golovin. Submodular function maximization. In Lucas Bordeaux, Youssef Hamadi, and Pushmeet Kohli (eds.), *Tractability: Practical Approaches to Hard Problems*, pp. 71–104. Cambridge University Press, 2014. doi: 10.1017/CBO9781139177801.004.

Sanjay Krishnan, Animesh Garg, Sachin Patil, Colin Lea, Gregory Hager, Pieter Abbeel, and Ken Goldberg. Transition state clustering: Unsupervised surgical trajectory segmentation for robot learning. *The International journal of robotics research*, 36(13-14):1595–1618, 2017.

Daehee Lee, Minjong Yoo, Woo Kyung Kim, Wonje Choi, and Honguk Woo. Incremental learning of retrievable skills for efficient continual task adaptation. *Advances in neural information processing systems*, 37:17286–17312, 2024.

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

Zuxin Liu, Jesse Zhang, Kavosh Asadi, Yao Liu, Ding Zhao, Shoham Sabach, and Rasool Fakoor. Tail: Task-specific adapters for imitation learning with large pretrained models. In *International Conference on Learning Representations*, volume 2024, pp. 16330–16353, 2024.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7765–7773. IEEE, 2018.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in neural information processing systems*, 32, 2019.

Ralf Römer, Yi Zhang, Yuming Li, and Angela P Schoellig. Clare: Continual learning for vision-language-action models via autonomous adapter routing and expansion. *IEEE Robotics and Automation Letters*, 2026.

Kaushik Roy, Akila Dissanayake, Brendan Tidd, and Peyman Moghadam. M2distill: Multi-modal distillation for lifelong imitation learning. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1429–1435. IEEE, 2025.

Tanmay Shankar and Abhinav Gupta. Learning robot skills with temporal variational inference. In *International Conference on Machine Learning*, pp. 8624–8633. PMLR, 2020.

Kyriacos Shiarlis, Markus Wulfmeier, Sasha Salter, Shimon Whiteson, and Ingmar Posner. Taco: Learning task decomposition via temporal alignment for control. In *International Conference on Machine Learning*, pp. 4654–4663. PMLR, 2018.

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

Weikang Wan, Jiawei Fu, Xiaodi Yuan, Yifeng Zhu, and Hao Su. Lodestar: Long-horizon dexterity via synthetic data augmentation from human demonstrations. In *9th Annual Conference on Robot Learning*, 2025.

Changti Wu, Bin Yu, Zhaolong Shen, Shijie Lian, Xiaopeng Lin, Cong Huang, Zhirui Zhang, Lei Zhang, and Kai Chen. Sieve: Structure-aware data selection for imitation learning with vla models. *arXiv preprint arXiv:2607.06442*, 2026.

Jingkai Xu and Xiangli Nie. Speci: Skill prompts based hierarchical continual imitation learning for robot manipulation. *IEEE Transactions on Cognitive and Developmental Systems*, 2025.

Mengda Xu, Zhenjia Xu, Cheng Chi, Manuela Veloso, and Shuran Song. Xskill: Cross embodiment skill discovery. In *Conference on robot learning*, pp. 3536–3555. PMLR, 2023.

Bin Yu, Shijie Lian, Xiaopeng Lin, Zhaolong Shen, Yuliang Wei, Changti Wu, Hang Yuan, Haishan Liu, Bailing Wang, Cong Huang, et al. Frameskip: Learning from fewer but more informative frames in vla training. *arXiv preprint arXiv:2605.13757*, 2026.

Hongquan Zhang, Jingyu Gong, Zhizhong Zhang, Xin Tan, Yanyun Qu, and Yuan Xie. T2s: Tokenized skill scaling for lifelong imitation learning. *arXiv preprint arXiv:2508.01167*, 2025a.

Yu Zhang, Yuqi Xie, Huihan Liu, Rutav Shah, Michael Wan, Linxi Fan, and Yuke Zhu. Scizor: A self-supervised approach to data curation for large-scale imitation learning. *arXiv preprint arXiv:2505.22626*, 2025b.

Fan Zhou and Chengtai Cao. Overcoming catastrophic forgetting in graph neural networks with experience replay. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 4714–4722, 2021.

Yifeng Zhu, Peter Stone, and Yuke Zhu. Bottom-up skill discovery from unsegmented demonstrations for long-horizon robot manipulation. *IEEE Robotics and Automation Letters*, 7(2):4126–4133, 2022.

A Implementation Details and Mathematical Properties

This appendix expands the task and replay formulation in Section 3.1 and specifies the construction in Sections 3.2–3.4. All structural labels, feature statistics, and selection scores are computed from the current task. A candidate is a training chunk indexed by its source demonstration and start time. Neighboring candidates may overlap, and each distinct chunk consumes one slot. The implementation uses $H = 10$ and $b = 1{,}000$ in the main LIBERO configuration. Full-budget construction requires at least b distinct candidates and, when anchors are required, at least one complete demonstration that fits within b.

A.1 Continual Imitation Learning and Replay Formulation

Tasks and training chunks. We consider a policy π_θ trained over manipulation tasks $T^1, \ldots, T^K$. At stage k, the current task provides expert demonstrations $D^k = \{\tau_1^k, \ldots, \tau_N^k\}$ containing time-ordered observation–action pairs and task instructions. The candidate set C^k comprises training chunks extracted from these demonstrations. A chunk c contains the sequence $(o_{c,1}, a_{c,1}), \ldots, (o_{c,H}, a_{c,H})$ and its source-task instruction ℓ_c, where $o_{c,t}$ and $a_{c,t}$ denote the recorded observation and expert action at position t within c. The policy uses the observation modalities and supervision format specified by its architecture. In particular, retaining a temporal chunk does not require every policy to consume an observation history of the same length. Episode-boundary handling follows the corresponding policy's native data loader. For BCTransformer, every recorded frame is a candidate start, with stride one and no extra frame stacking. All n_j starts of a demonstration of length n_j are retained as candidates, including starts with fewer than $H = 10$ remaining positions. The loader repeats the final observation and action to complete these tail chunks. A complete-demonstration anchor therefore costs n_j chunk slots, rather than the number of disjoint ten-step segments. VLA-specific candidate boundaries and horizons are specified in Table 7.

Policy-specific imitation objectives. Let $\ell_{\mathrm{BC}}(\theta; c)$ denote the native imitation-learning loss associated with candidate c. For BCTransformer, the policy predicts each action from the instruction and the causal observation history $h_{c,t} = (o_{c,1}, \ldots, o_{c,t})$ within the chunk. Its chunk loss averages the negative log-likelihood over the H supervised positions:

$$\ell_{\mathrm{BC}}(\theta; c) = -\frac{1}{H} \sum_{t=1}^{H} \log \pi_\theta(a_{c,t} \mid h_{c,t}, \ell_c). \tag{6}$$

Here $h_{c,t}$ is an observation sequence, while $a_{c,t}$ is a single expert action. For Qwen-GR00T and Qwen-OFT, ℓ_{BC} instead denotes the native flow-matching loss and mean L1 action error, respectively, with their original policy inputs and prediction horizons. HMR changes the retained replay data without changing these policy-specific objectives.

Per-task memory construction and replay. After training on task k, the selector retains $S^k \subseteq C^k$ with $|S^k| = b$ and updates memory as

$$M_k = M_{k-1} \cup S^k, \qquad M_0 = \varnothing. \tag{7}$$

Each candidate is identified by its source task, demonstration, and start time. Distinct overlapping chunks therefore consume separate slots, including chunks from complete-demonstration anchors. Each full-quota update adds b chunks; previously retained task selections remain fixed. Memory construction may be omitted after the final task when no subsequent replay uses it.

During training on task k, policy-training samples come from C^k and the historical memory M_{k-1}. Their relative contributions are expressed by

$$L_{\mathrm{ER},k}(\theta) = \mathbb{E}_{c \sim C^k}[\ell_{\mathrm{BC}}(\theta; c)] + \lambda \mathbb{E}_{c \sim M_{k-1}}[\ell_{\mathrm{BC}}(\theta; c)], \tag{8}$$

where $\lambda \geq 0$ controls the relative replay contribution. The expectations are estimated with current-task and historical minibatches, and the replay term is omitted for the first task. This expression specifies relative weighting; the numerical loss reduction follows the native implementation. For example, averaging the concatenation of equal-sized current-task and replay batches gives equal relative weight to the two sources, with a common normalization factor. Architecture-specific inputs, losses, batch sizes, and sampling protocols are given in Sections B.1 and B.2.

A.2 Temporal Behavioral Structure Discovery

Chunk descriptors and local distance. For actions, gripper states, and joint states, we concatenate each value with its first temporal difference. The initial difference is zero. A chunk descriptor contains the resulting sequence over its H positions; positions beyond an episode boundary repeat the boundary value. No extra context is appended to this descriptor. Each coordinate is centered by its median and scaled by 1.4826 times its median absolute deviation over all chunk positions in the task. A near-zero scale falls back to the standard deviation and then to one. Values are clipped to $[-10, 10]$, and the action, gripper, and joint blocks are weighted by 1, 2, and 0.5, respectively.

Let $x_1, \ldots, x_H$ and $y_1, \ldots, y_H$ be the two chunk descriptors, with d coordinates at each position. Their local matching cost is $\delta(x_i, y_j) = \|x_i - y_j\|_2 / \sqrt{d}$. Let Π_w contain paths from $(1,1)$ to (H,H) with steps $(1,0)$, $(0,1)$, or $(1,1)$, restricted to $|i-j| \leq w$, where $w = \max\{1, \lceil 0.15H \rceil\}$. The implementation computes

$$\begin{aligned} P^\star &\in \arg\min_{P \in \Pi_w} \sum_{(i,j) \in P} \delta(x_i, y_j), \\ D_{\mathrm{DTW}}(c, c') &= \frac{1}{|P^\star|} \sum_{(i,j) \in P^\star} \delta(x_i, y_j). \end{aligned} \tag{9}$$

Dynamic programming minimizes accumulated cost. Ties between predecessor costs prefer the diagonal predecessor, followed by the previous position in the first sequence and then the previous position in the second. The selected path's accumulated cost is divided by its number of matched position pairs. Minimizing average path cost directly would define a different selection rule.

Progress alignment. The progress prototype is a reference feature sequence estimated separately for each task. Let $f_{j,t} \in \mathbb{R}^{d_f}$ be the frame descriptor at zero-based time t in demonstration j, which contains n_j frames. It concatenates low-dimensional values and their first differences. Each action, gripper, or joint block is standardized over the task, divided by the square root of its dimension, and weighted by 1, 2, or 0.5, respectively. These frame descriptors use the progress-alignment normalization and are distinct from the robustly standardized chunk descriptors used for local matching above.

Initialization. The prototype has $P = 48$ positions indexed from zero to $P-1$. Each demonstration is linearly interpolated at the uniformly spaced positions $p(n_j-1)/(P-1)$. Denoting the interpolated features by $\widetilde{f}_{j,p}$, we initialize

$$\mu_p^{(0)} = \frac{1}{N}\sum_{j=1}^{N} \widetilde{f}_{j,p}. \tag{10}$$

A one-frame demonstration repeats its sole descriptor at every prototype position. Interpolation is coordinatewise between adjacent frames and includes both episode endpoints.

Alignment and update. At iteration u, aligning frame t to prototype position p has cost

$$E_j^{(u)}(t,p) = \frac{\|f_{j,t} - \mu_p^{(u)}\|_2^2}{d_f} + \beta\left(\frac{t}{\max\{1, n_j-1\}} - \frac{p}{P-1}\right)^2, \qquad \beta = 0.25. \tag{11}$$

For each demonstration, DTW finds a path $\Gamma_j^{(u)}$ minimizing the sum of these costs from $(0,0)$ to $(n_j-1, P-1)$, with the same three monotone moves as above. This alignment searches the full frame–prototype grid; the progress penalty regulates temporal displacement without imposing the local chunk matcher's hard band. Predecessor ties prefer diagonal, then previous frame, then previous prototype position. The returned diagnostic alignment cost is the accumulated cost divided by path length; prototype updates use the path itself.

Let $A_{j,p}^{(u)}$ be the set of distinct frame indices t for which $(t,p) \in \Gamma_j^{(u)}$. The prototype update is

$$\mu_p^{(u+1)} = \frac{1}{N}\sum_{j=1}^{N}\left(\frac{1}{|A_{j,p}^{(u)}|}\sum_{t \in A_{j,p}^{(u)}} f_{j,t}\right). \tag{12}$$

Endpoint-covering paths assign frames to every prototype position for nonempty demonstrations. The implementation also retains the previous prototype value if a position receives no contribution. Averaging within a demonstration before averaging across demonstrations gives every demonstration equal weight at each prototype position, regardless of how many frames it contributes there. Two alignment–update iterations are performed, yielding $\mu^{(2)}$.

Final progress assignment. All demonstrations are aligned once more to $\mu^{(2)}$, without a further prototype update. If Γ_j^{final} is the resulting path, the progress of frame t is

$$\rho_j(t) = \text{mean}_{p:\,(t,p)\in\Gamma_j^{\text{final}}}\, p. \tag{13}$$

Thus a frame matched to several prototype positions receives their mean, which may be noninteger. A chunk c starting at frame t_c in demonstration j inherits $\rho(c) = \rho_j(t_c)$ from Equation 13. Rounding is applied when forming the occurrence histogram. These progress coordinates organize memory selection within the task and are not additional inputs to the policy. Global alignment locates corresponding execution stages, while local DTW measures behavioral similarity for mode clustering.

Medoids and the number of modes. At most 1,000 candidates are sampled across demonstrations and eight progress strata. Candidates are partitioned by demonstration and progress stratum, using eight equal-width intervals between zero and the largest observed progress coordinate. Within each nonempty partition, a seeded random permutation is formed; candidates are then drawn round-robin in sorted partition order, skipping exhausted partitions, until the pool reaches its size limit. All candidates are used when their number does not exceed the limit. Medoid initialization starts with the candidate of minimum total distance to the pool, followed by farthest-point additions. Up to 20

iterations alternate nearest-medoid assignment and within-cluster minimum-total-distance updates; an empty cluster is reinitialized from an unselected candidate farthest from the existing medoids.

Let B_{ref} be the post-anchor capacity under the equal-weight reference ranking, and let n_{pool} be the discovery pool size. The maximum tested mode count is the smaller of n_{pool} and $\max\{1, \lfloor B_{\text{ref}}/20 \rfloor\}$. We test the feasible entries of $\{5, 10, 20, 30, 40\}$ together with this maximum. Each clustering produces occurrence sets for individual demonstrations. We choose the clustering maximizing

$$(1-J)\log(1+R), \tag{14}$$

where J is the median pairwise Jaccard similarity between these sets and R is the median distinct-demonstration support of an occurrence. Ties favor fewer modes. The implementation also computes similarity of collapsed occurrence sequences using normalized longest common subsequence; its minimum admissible value is zero in this configuration, so it does not exclude any candidate clustering. This mode-count rule is a selection heuristic. All task chunks are subsequently assigned to the nearest medoid.

Occurrences and transitions. For each mode, we histogram rounded progress coordinates and smooth the histogram with a normalized Gaussian kernel of radius two and standard deviation $4/3$ progress positions, using zero extension at the histogram boundaries. Local maxima must reach 35% of the largest smoothed value and be separated by at least five phase positions; at most four peaks are retained before support filtering. Each peak needs support within two phase positions from at least the larger of two demonstrations and 10% of the demonstrations represented in the discovery pool, rounded up. If none survives, a single occurrence is placed at the median progress of that mode. Chunks are assigned to the nearest occurrence center of their mode.

Along each demonstration, a change between successive valid group labels creates a directed transition. Its boundary is the start of the first chunk in the new group. A chunk supports the transition exactly when its span contains this boundary; it need not contain a complete execution of both groups. Repeated instances of a transition type count only once toward its support from a given demonstration. Final coverage eligibility is recomputed over all task candidates. Let U_j collect occurrence labels and directed transition types from all candidate chunks of demonstration j, and let $U = \bigcup_{j=1}^{N} U_j$. Their cross-demonstration support and the reliable coverage targets are

$$\text{supp}(u) = \sum_{j=1}^{N} \mathbf{1}[u \in U_j], \qquad U_{\text{rel}} = \{u \in U : \text{supp}(u) \geq n_{\min}\}. \tag{15}$$

The active configuration uses $n_{\min} = \max\{2, \lceil 0.1N \rceil\}$. Each demonstration contributes at most once for each occurrence or transition type. Fallback occurrences undergo the same eligibility test. This threshold determines which structures are eligible for coverage; the support targets κ_v specify how much support selected memory should provide for those structures.

Dense importance. The dense signal uses neither policy embeddings nor BC losses. Within each demonstration, action changes, changes in gripper commands and states, and joint changes are converted into positive robust deviations from their medians, using a MAD scale with standard-deviation fallback and clipping to $[0, 8]$. Their weighted sum, with weights 1, 2, and 0.5, gives eventness after percentile scaling. Eventness is set to zero outside normalized progress $[0.05, 0.98]$.

At each aligned phase, consensus quality combines the fraction of demonstrations with eventness at least 0.5, mean eventness, and feature compactness around the prototype, with weights 0.45, 0.25, and 0.30. A floor of 0.05 keeps this quality positive. Compactness decays exponentially with mean prototype distance, using the median positive alignment distance as scale. A candidate's consensus affinity is the maximum, within two nearby phases, of phase quality times Gaussian feature and progress affinities; the progress scale is 1.5. Feature affinity uses the phase median squared distance plus 0.1 times the global distance scale and a numerical stabilizer.

A boundary score averages the scaled feature change across a radius-four temporal neighborhood and the corresponding prototype gradient. The raw importance is the product of consensus affinity and $(0.2 + 0.5\,\text{eventness} + 0.3\,\text{boundary})$. After scaling, this score is propagated within each demonstration by exponential maximum propagation with decay scales six toward earlier positions and twelve toward later positions. The final signal is the scaled maximum of the local score and 0.8

times its propagated value. Scaling in this dense module maps the 10th and 90th percentiles to zero and one, clips outside this interval, and returns zero for a degenerate interval. This differs from the min–max scaling used in memory completion below.

A.3 Anchor Rankings, Stability, and Packing

Event and central-diverse rankings. The event ranking uses a dense chunk shortlist of size $\min\{|C^k|, \lceil 1.5b \rceil\}$. A sequential relevance–redundancy selector constructs this shortlist from the highest-importance candidates, using a pool up to four times the shortlist size. It subtracts maximum positive cosine similarity to previously selected descriptors and maximum same-demonstration temporal overlap, both with weight 0.05; temporal overlap decays with scale eight. Descriptors are reduced to at most 24 principal components when necessary and normalized to unit length. Demonstrations are ranked by descending shortlist chunk count, then earliest shortlist position, descending summed importance, shorter length, and identifier.

The central-diverse ranking summarizes each action, gripper, and joint coordinate by its mean, standard deviation, first and last values, and the mean and standard deviation of first differences. These summaries are standardized across demonstrations and normalized per trajectory. When RGB views are available, the descriptor additionally uses eight uniformly sampled frames per view, represented by mean RGB values on a 4×4 grid. Up to two available views are used; the standardized, normalized visual block and low-dimensional block are combined with equal norm. The ranking begins nearest the descriptor mean, followed by largest minimum distance to the already ranked demonstrations.

The combined ranking averages zero-based ranks normalized by $\max\{1, N-1\}$; ties use event rank, central-diverse rank, and identifier. Each ranking is packed into complete demonstrations before comparison.

Reference structure and comparison. The equal-weight ranking supplies a reference anchor pack and discovery budget. The structure discovered with that budget is shared by the endpoint comparisons and retained for final coverage and completion. Bootstrap-like subsampling below constructs temporary alternative structures only for testing the ranking preference; these structures do not replace the reference labels in the final memory.

For each candidate anchor pack, we count covered reliable modes, occurrences, and transition types, requiring at least two source demonstrations for reliability in this comparison. A separate greedy probe then adds chunks that cover the largest number of still-uncovered types, breaking ties by dense importance and chunk index. It records remaining uncovered types and the post-anchor budget minus the number of probe chunks. The probe uses binary type coverage, includes modes, and does not impose the final coverage ceiling or distinct-demonstration targets. It is a diagnostic cost estimate, not the coverage pass described below, and its greedy cost is not asserted to be a mathematical lower bound.

An endpoint wins if it has no smaller coverage of each of the three type categories, no larger remaining uncovered-type count, and no smaller post-probe capacity, with strict improvement in at least one criterion. Otherwise the comparison has no winner.

Subsampling stability. We request ten trials, each sampling $\lceil 0.8N \rceil$ demonstrations without replacement, with a minimum of two when available. Each trial re-estimates the progress prototype and mode/occurrence discovery using the sampled demonstrations, reassigns all task chunks to that temporary structure, and compares the fixed candidate anchor packs. Let L count completed trials and w_l denote the outcome of trial l, indicating a preference for either ranking or no preference. For $L > 0$, the empirical support for ranking a is

$$s(a) = \frac{1}{L} \sum_{l=1}^{L} \mathbf{1}[w_l = a]. \tag{16}$$

Completed trials with no preference remain in the denominator. A ranking is adopted only if at least five trials complete and its support satisfies $s(a) \geq \tau$, with $\tau = 0.8$. Otherwise the equal-weight ranking is used, including when no trial completes. Trial sampling uses the run seed plus a fixed

offset of 7,919. The frequency is a measure of selection stability, not a confidence bound on policy performance.

Complete-demonstration packing. The target interval and demonstration-count limits are given in Table 8, with ablation overrides in Table 9. Proportion targets are rounded to chunk counts, and the demonstration-count limits are enforced throughout packing. We first add ranked complete demonstrations that fit within the total quota until the minimum demonstration count is met. While below the lower target, the first ranked demonstration that fits the upper target is added. If none does, the fitting demonstration with the smallest overshoot above the lower target is added. A final ranked pass adds demonstrations that still fit the upper target. The overall quota b is never exceeded, but the soft interval may be missed because episodes are indivisible. If anchors are enabled and no complete episode fits, construction raises an error.

A.4 Structural Coverage and Its Set-Function Property

For each reliable occurrence or transition in U_{rel}, the active configuration creates a chunk-count target of one and a distinct-demonstration target of two. These are separate requirements in V^k; consequently one chunk can supply gain to more than one requirement. Counts include all anchor chunks. The auxiliary per-demonstration and coarse-progress coverage targets are disabled. Only unselected current-task chunks are eligible for addition. Starting from the anchor set $S = A^k$, the marginal gain of candidate c is

$$\Delta_{\text{cov}}(c \mid S) = F_{\text{cov}}(S \cup \{c\}) - F_{\text{cov}}(S). \tag{17}$$

The coverage cap is the smaller of the remaining candidate count and $\text{round}(0.2(b - |A^k|))$. At each step, candidates are first restricted to start times at least five steps from every selected start in the same demonstration. Among positive-gain candidates, selection maximizes Equation 17; ties are resolved by transition chunk gain, transition demonstration gain, occurrence chunk gain, occurrence demonstration gain, centrality, dense importance, and smaller chunk index, in that order. If no positive-gain candidate passes the gap, it is relaxed to two and then zero. Selection stops when the cap is reached or no positive gain remains at zero gap. Unmet targets are recorded and do not trigger reallocation of the protected subset. Coverage is best effort when its cap is insufficient.

Proposition 1: diminishing returns of structural support. Fix the candidate set, each candidate's structural incidences and source demonstration, and all nonnegative targets κ_v. Then F_{cov} in Equation 3 is normalized, monotone, and submodular.

Proof. A chunk-count requirement has the form $n_v(S) = \sum_{c \in S} \mathbf{1}[c \text{ supports } v]$. A distinct-demonstration requirement counts the size of the union of singleton source-demonstration sets contributed by chunks supporting v. Both counts are monotone, vanish at the empty set, and have diminishing marginal increments: an additional supporting chunk contributes one to the first count, and contributes one to the second only if its source demonstration is not yet represented for that requirement.

For $X \subseteq Y$ and $c \notin Y$, the count increment at X is therefore at least the increment at Y, while the remaining capacity before reaching κ_v is also at least as large at X. Capping the count at κ_v preserves this ordering of marginal increments. Summing over requirements gives

$$F_{\text{cov}}(X \cup \{c\}) - F_{\text{cov}}(X) \geq F_{\text{cov}}(Y \cup \{c\}) - F_{\text{cov}}(Y). \tag{18}$$

Normalization and monotonicity follow term by term. With fixed anchors, the residual function $F_{\text{cov}}(A^k \cup R) - F_{\text{cov}}(A^k)$ has the same properties on non-anchor candidates. This is an instance of the diminishing-returns framework discussed by Krause & Golovin (2014). □

The proposition concerns the fixed support objective. The implementation uses state-dependent temporal eligibility and gap relaxation, so we do not claim the usual pure-cardinality greedy approximation factor for this operational procedure. The proof also gives no guarantee that a count target is sufficient for successful policy execution.

A.5 Representative Completion

Group allocation. Group frequency N_g counts all source-task chunks in the group. Importance is the 75th percentile of dense scores over that same full group, including already retained chunks, followed by min–max normalization across groups. The current configuration uses linear frequency weighting, $\eta = 0.25$, and $\alpha = 1$. For p_g we count chunks in P^k; the group capacity Q_g counts its unselected candidates. A slot is assigned to the largest-priority group with unfilled capacity; priority ties prefer larger source count and then smaller mode and occurrence identifiers. Allocations are fixed before representative chunk selection.

Normalization and score components. For the completion module, min–max scaling is computed over finite values. A nonconstant finite vector is mapped to $[0,1]$; a constant finite vector maps to one, and entries without a finite value map to zero. The inverted version applies one minus this result on finite entries. Thus s_{cen} is the inverted scaling of distance to the assigned mode medoid over all task chunks, whereas s_{evt} is the scaling of the dense scores over all task chunks. Behavioral groups with the same mode share this mode medoid; no separate medoid is fitted for each occurrence. Representative initialization searches within the group's eligible candidates using distance to its assigned mode medoid.

For diversity, the sequence descriptors defined in Appendix A.2 are flattened, and each coordinate is divided by its task-wide standard deviation, using one for a scale below 10^{-8}. Let $x(c)$ be this vector. The unnormalized diversity is

$$d_{\text{raw}}(c,S) = \min_{s\in S} \|x(c) - x(s)\|_2. \tag{19}$$

Its min–max scaling is recomputed over the remaining candidates in groups with unfilled quotas, before temporal-gap filtering, to obtain $d(c,S)$. The implementation maintains nearest-selected distances incrementally. At least one complete anchor demonstration ensures the initial set is nonempty in the stated configuration.

Let $n_{\text{demo}}(c,S)$ count selected chunks from the source demonstration of c, and let $\delta(c,S)$ be the smallest absolute start-time difference to such a chunk. The two repetition terms are

$$g(c,S) = \frac{1}{1+n_{\text{demo}}(c,S)}, \qquad o(c,S) = \exp\bigl(-\delta(c,S)/8\bigr). \tag{20}$$

If no selected chunk shares the demonstration, $o(c,S) = 0$. The weights in Equation 5 are $(w_{\text{cen}}, w_{\text{evt}}, w_{\text{div}}, w_{\text{demo}}, w_{\text{time}}) = (0.5, 0.5, 0.25, 0.25, 0.25)$. No policy-loss or policy-latent tie-breaker is used for this core selection.

Filtering and selection order. Within each group, we compute the 99.5th percentile of valid distances to its mode medoid over unselected candidates. Candidates above the threshold are discarded only if the survivors can still fill the entire group allocation; otherwise, all unselected candidates are restored. Groups with positive allocations first contribute one representative, processed in sorted group order. At the first feasible temporal gap in $\{5,2,0\}$, the representative maximizes centrality, then importance, then prefers smaller index.

After these representatives, each iteration considers remaining candidates from all groups with unfilled quotas. It uses the first of the same three gap levels admitting a candidate, selects the largest dynamic score, and breaks ties by smaller index. Counts, nearest distances, and temporal neighbors are updated after every addition. Gap relaxation affects the candidate's eligibility; it does not remove the temporal penalty from its score. Selection stops only when the prescribed additional budget is filled. Insufficient initial capacity or unexpected candidate exhaustion raises an error instead of silently returning an underfilled memory.

A.6 Discrete Utility of Group Allocation

The allocation rule admits an exact utility interpretation. Write $W_g = N_g(1+\eta I_g)$ and, for a nonnegative integer a, define

$$\Phi_g(a) = W_g \sum_{j=0}^{a-1} \frac{1}{\alpha + p_g + j}, \qquad \Phi_g(0) = 0. \tag{21}$$

The empty sum is zero. Its next-unit increment is

$$\Phi_g(a+1) - \Phi_g(a) = \frac{W_g}{\alpha + p_g + a}. \tag{22}$$

This is exactly the implemented priority, including the support already provided by anchors and coverage. It is a discrete utility; identifying the priority with a derivative of a logarithm would not by itself establish this exact finite-increment equivalence.

Proposition 2: optimality for the allocation subproblem. Fix groups, $W_g \geq 0$, $p_g \geq 0$, $\alpha > 0$, integer candidate capacities Q_g, and a feasible integer budget b^k_{rem}. Repeatedly assigning the next slot to the feasible group of largest priority in Equation 4 solves

$$\begin{aligned} \max_{\{a_g\}} \quad & \sum_g \Phi_g(a_g) \\ \text{subject to} \quad & \sum_g a_g = b^k_{\text{rem}}, \qquad 0 \leq a_g \leq Q_g, \quad a_g \in \mathbb{Z}. \end{aligned} \tag{23}$$

Proof. Each group supplies a nonincreasing sequence of marginal values $W_g/(\alpha + p_g + j)$ for $j = 0, \ldots, Q_g - 1$. An allocation selects a prefix of this sequence for each group. At any iteration, the first unselected value in a group is at least as large as every later value in that group. The largest feasible next value is therefore a largest value among all remaining marginal values. Selecting it preserves the prefix property. Repeating for b^k_{rem} steps yields a feasible selection of the largest b^k_{rem} marginal values, with ties resolved consistently with prefixes. No other allocation can have a greater sum. This is the separable diminishing-marginal-utility case of discrete resource allocation (Federgruen & Groenevelt, 1986). □

This result holds only for fixed group allocation. The subsequent chunk score changes with selected descriptors and dynamic normalization; its diversity and repetition terms are not asserted to be increments of $\sum_g \Phi_g$. Neither proposition establishes optimal replay composition for BC loss or a bound on forgetting.

A.7 Construction Order and Implementation Scope

For clarity, the construction proceeds in the following order.

1. Extract all current-task chunks, sequence descriptors, progress alignments, and dense importance. Build the event and central-diverse demonstration rankings.
2. Pack the equal-weight ranking and discover the reference modes, occurrences, and transitions under its post-anchor budget.
3. Compare endpoint anchor packs under reference and subsampled structures, choose the stable endpoint or the equal-weight fallback, and retain all chunks from the resulting complete demonstrations.
4. Starting from those anchors, add positive-gain coverage chunks until the support targets or coverage stopping conditions are reached. Freeze their union as P^k.
5. Count source-group frequency, importance, retained support, and remaining capacity; allocate exactly $b - |P^k|$ additional slots.
6. Initialize group representatives and fill the allocations using the shared dynamic score, yielding the disjoint set R^k.
7. Return $S^k = P^k \cup R^k$ and append it to replay memory after task training, as in Equation 7.

All selected chunks retain their source-task instruction. Existing tasks' retained chunks are not reallocated by this core construction. The mathematical properties above are conditional on the discovered structure; they do not assert that the discovery heuristic is optimal. In the reported comparisons, retained-support conditioning refers to chunks already selected for the current task. The reported replay selections follow the anchor–coverage–completion construction; historical-memory replacements do not contribute to these selections.

Table 2: **Policy architectures and training settings on LIBERO.** Policy-training settings are shared across memory-selection methods within each architecture.

Setting	BCTransformer	Qwen-GR00T
Visual/language model	ResNet-18; fixed BERT task embeddings	Qwen3-VL-4B-Instruct
Initialization	Policy trained from scratch	Pretrained VLM; new action head
Policy inputs	Two RGB views, joints, gripper, instruction	Two RGB views and instruction
Temporal observation input	Up to 10 observations	Current observation
Action supervision	Per-position action labels over 10-step chunks	8 future actions per input
Action head	Five-component GMM	GR00T DiT-B
Training loss	Mean action negative log-likelihood	Flow matching
BC RGB resolution	128×128	—
BC image augmentation	Translation: 8; brightness/contrast/saturation/hue jitter: 0.3 each	—
BC temporal Transformer	4 layers; 6 heads; head dimension 64; MLP 256; dropout 0.1	—
Parameter updates	Full policy	Full VLM and action head
Training per task	50 epochs	10,000 optimizer updates
Current/replay batch size	64 / 64	8 / 8
Gradient accumulation	1	1
Optimizer	AdamW	AdamW
Adam coefficients	$(0.9, 0.999)$	$(0.9, 0.95)$
Initial learning rate	10^{-4}	VLM: 10^{-5}; action head: 10^{-4}
Default LR for other parameters	10^{-4}	2.5×10^{-5}
Learning-rate schedule	Cosine decay	Cosine decay with 100 warmup updates
Minimum learning rate	10^{-5}	10^{-6}
Weight decay	10^{-4}	10^{-8}
Gradient-norm clipping	100	1
Optimizer/scheduler at task boundary	Reinitialized	Reinitialized

B Experimental Protocol and Metric Definitions

B.1 Model, Training, and Evaluation Settings

Tables 2 and 3 summarize the LIBERO configurations for the two policy architectures. Methods within each architecture share the policy-training and replay budgets. The BCTransformer experiments use the full current-task demonstration set for training; selection restricts later replay. The Qwen-GR00T experiments preserve the model's native flow-matching loss and eight-action prediction horizon.

Both protocols omit replay before any earlier task is available. Memory indices remain fixed after selection, and replay sampling is proportional to each task's retained count. Overlapping chunks count as distinct training samples, so equal chunk quotas do not imply equal numbers of unique frames or identical storage across architectures. The deferred BCTransformer selectors may omit memory construction after the final task, when no subsequent replay uses it.

Qwen-GR00T adaptation. The cross-architecture comparison uses ER and the core construction of HMR. ER retains the first 1,000 valid action-chunk starts in episode/frame order. Our selector uses eight-step chunks and the available end-effector pose as a proxy for the unavailable joint-angle

Table 3: **Replay and evaluation protocols on LIBERO.** S/O/G/L denote the Spatial, Object, Goal, and Long suites, respectively.

Setting	BCTransformer	Qwen-GR00T
Task stream	10 tasks per suite; order 0	10 tasks per suite; order 0
Replay quota	1,000 chunks per task	1,000 action-chunk starts per task
Historical sampling	Uniform over retained chunks	Uniform over retained starts
Checkpoint evaluated	Epochs $0, 5, \ldots, 50$ on current task	Final 10,000-update checkpoint only
Checkpoint carried forward	Highest current-task success; earliest tie	Final checkpoint
Retention evaluation	Stage checkpoints on all seen tasks	Stage checkpoints on all seen tasks
Rollouts per checkpoint/task	20	50
Maximum action steps	600 for all suites	S: 220; O: 280; G: 300; L: 520
Training seeds	10000, 10001, 10002	42, 10000
Qwen-GR00T evaluation seed	—	42 for all training seeds
Reported metrics	FWT, NBT, AUC	Final ASR, NBT, stage-based AUC

Table 4: **Training demonstrations in the Qwen-GR00T LIBERO conversions.** Columns index successive tasks in the benchmark's predefined order (order index 0). The counts are shared by ER and HMR and are distinct from the 50 demonstrations per task used in the BCTransformer experiments.

Suite	1	2	3	4	5	6	7	8	9	10
Spatial	45	45	46	43	42	39	46	35	47	44
Object	44	45	47	46	45	42	45	45	50	45
Goal	43	48	47	36	46	33	40	50	49	36
Long	33	49	41	35	38	41	36	43	29	34

channels. Its complete-demonstration ranking uses low-dimensional trajectory descriptors in this adapter. Both methods use the same LeRobot conversion and fixed suite-level normalization statistics; these statistics may include future-task data, while policy-training samples remain restricted to the current task and retained memory. Each stage's final policy initializes the next stage.

The source LIBERO datasets provide 50 demonstrations per task, all of which are used by BCTransformer. The deployed LeRobot conversions used by Qwen-GR00T contain the episode counts in Table 4. ER and HMR use the same converted episodes within this comparison. These counts describe the available training data before memory selection; they are not the number of demonstrations selected for replay.

B.2 RoboTwin-CL Configuration

RoboTwin-CL comprises the eight tasks in Table 5, presented in the listed order. We use the clean end-effector-control demonstrations and train StarVLA's Qwen-OFT (Community, 2026) with the same task sequence and training protocol for ER and HMR. Each task contains 50 clean demonstrations, for a total of 400. The data use the deployed LeRobot end-effector conversion of the `demo_clean_collect_200` collection, with 50 episodes per task. Table 6 summarizes the configuration.

Qwen-OFT applies an MLP to the final-layer hidden states of action tokens and predicts continuous actions in parallel. The loss averages absolute action errors over the concatenated current-task and replay batch, giving equal weight to the two sources when their batch sizes match. The first task initializes the VLM from pretrained weights and the action head from scratch; each subsequent task inherits the preceding task's final policy.

Action normalization uses fixed minima and maxima pooled over the eight tasks' training demonstrations, shared across methods and stages. Non-gripper coordinates are mapped to $[-1, 1]$, while the two gripper channels are binarized at 0.49. Thus preprocessing uses future-task training ranges,

Table 5: **Task sequence for RoboTwin-CL.** All methods learn the eight tasks in ascending stage order. Task names correspond to RoboTwin environment identifiers.

Stage	Task	Stage	Task
1	`adjust_bottle`	5	`click_bell`
2	`grab_roller`	6	`place_container_plate`
3	`beat_block_hammer`	7	`shake_bottle`
4	`lift_pot`	8	`stack_bowls_two`

Table 6: **Qwen-OFT configuration on RoboTwin-CL.** Model, training, replay, and evaluation settings are shared by HMR and ER.

Setting	Qwen-OFT on RoboTwin-CL
Pretrained backbone	Qwen3-VL-4B-Instruct
Action-head initialization	Newly initialized MLP
Policy inputs	Three current RGB views and task instruction; no proprioception
Cameras and image resolution	Head, left wrist, right wrist; 224×224 each
Action-token hidden dimension	2560
Regression head	Two residual MLP blocks, hidden dimension 5120
Action representation	16-dimensional absolute end-effector actions
Prediction / execution horizon	50 / 50 actions
Training loss	Mean L1 action error
Parameter updates	Full VLM and action head
Optimizer updates per task	10,000
Current / replay batch size	8 / 8; no replay for the first task
Gradient accumulation	1
Optimizer	AdamW; coefficients $(0.9, 0.95)$; epsilon 10^{-8}
Learning rate	VLM: 10^{-5}; action head: 10^{-4}
Default LR for other parameters	10^{-5}
Learning-rate schedule	Cosine decay; 5,000 warmup updates per task
Minimum learning rate	5×10^{-7}
Weight decay / gradient clipping	10^{-8} / norm 1
Training precision	BF16
Task boundary	Inherit final policy; reinitialize optimizer and scheduler
Replay quota	1,000 chunk starts per task; at most 7,000 historical starts
Historical sampling	Shuffled passes through the union of retained chunks
Evaluated checkpoints	Final checkpoint of each task on all seen tasks
Rollouts per checkpoint/task	25
Environment setting	Clean setting with unseen language instructions
Evaluation seed bank	Initial seed 100000; fixed task-specific seed banks
Rollout termination	Success or the environment's task-specific step limit
Training seeds	42, 10000
Reported metrics	Final ASR, NBT, stage-based AUC

although policy-training batches contain only current-task data and retained historical chunks. Action chunks respect episode and instruction boundaries and repeat the final action when padding is needed. Predicted quaternions are normalized before execution.

Evaluation is deferred until the task stream has finished, then evaluates the final checkpoint of each stage on every task seen by that stage. This yields 36 checkpoint–task pairs for the eight-task stream. All methods reuse fixed task-specific evaluation seed banks after filtering for expert-solvable initializations. We omit memory maintenance after the final task, since there is no subsequent replay.

B.3 Baseline Adaptations

For the BCTransformer comparison, all selectors determine which current-task chunks become available for later replay. They leave the BC training objective and previously committed memory unchanged. GSS uses the selected current-task policy checkpoint for scoring. Its auxiliary computations do not update the policy carried into the next task. The following adaptations specify selection timing, representation, and conversion to the common chunk budget.

ER. The benchmark implementation retains the first b chunks in each task's dataset order. Replay then shuffles the concatenation of these retained subsets. Thus, the reported ER reference uses a deterministic prefix selection rule; randomness enters replay sampling and policy training.

GSS. We adapt GSS-Greedy (Aljundi et al., 2019b) to a single pass over a random permutation of current-task chunks after policy training. The policy remains fixed, and gradients are computed from the same chunk BC loss over all trainable policy parameters. Candidate and reference batch sizes are both 10, with up to ten reference batches drawn from the temporary current-task memory. After the pool reaches b chunks, a candidate batch is considered for replacement only when its maximum cosine similarity to the reference gradients is below zero. The selector then applies the Greedy similarity-based probabilistic replacement rule. It replaces entries only in the current-task pool and performs no policy optimization during selection. This changes the original online admission schedule and global-buffer lifecycle to match our task-end, fixed-per-task-budget setting.

DemInf. We use the source observation and action β-VAEs and KSG mutual-information estimator (Hejna et al., 2025). The deployed configuration trains one observation VAE and one action VAE per suite and seed, with latent dimensions 16 and 6, respectively, and $\beta = 0.01$. Each VAE receives 50,000 updates with batch size 512. Normalization statistics and VAE training data are pooled across all ten tasks in the suite. The observation encoder uses agent-view images and end-effector/gripper state, following the adapted source input format. Frozen embeddings are scored separately within each task using neighbor counts 5, 6, and 7, scoring batches of 1,024, and four scoring repeats. Scores are clipped at the task's first and 99th percentiles, standardized, and averaged within each demonstration.

Demonstrations are ranked by decreasing score. The selector first considers those at or above the median score, extending to lower-ranked demonstrations only if needed to meet the budget. It retains all chunks of each chosen demonstration, except that the last demonstration contributes only the temporal prefix required to reach b chunks. The auxiliary models are trained before continual policy learning, but each task's selected chunks become replay memory only after that task has been learned. Because the shared representation and normalization use future-task demonstrations, this suite-wide variant receives additional information beyond the observed-task setting of HMR and GSS. Equal policy-training and replay budgets do not imply equal auxiliary data access or selection compute.

ER on RoboTwin-CL. The RoboTwin-CL ER baseline retains the first 1,000 valid chunk starts in the task's dataset order, or all available chunks if fewer than 1,000 are available. The selected indices remain fixed for later replay. ER uses the same training, normalization, and replay sampling as HMR; only memory selection differs.

B.4 Memory Construction Settings

Selection inputs across architectures. Table 7 specifies the signals used by the memory selector. These differ from the policy inputs: low-dimensional states can guide memory selection even when the policy itself consumes only images and language. The listed action and state channels, together with their first temporal differences, provide the descriptors for progress alignment and local DTW. Visual summaries are used only for central-diverse anchor ranking. Selection-specific feature normalization follows Section A.2 independently of the policy's action normalization.

For Qwen-GR00T, intact episodes include starts near their natural end and use endpoint padding. If decoding requires removal of a video suffix, starts whose eight-step action span crosses the cut are excluded. For Qwen-OFT, each instruction segment is a separate selection sequence; chunks stay

Table 7: **Architecture-specific inputs to memory construction.** Each retained index identifies a native training sample. All adapters enumerate valid starts at stride one; the selector horizon does not imply that the policy consumes an equally long observation history.

Selection setting	BCTransformer / LIBERO	Qwen-GR00T / LIBERO	Qwen-OFT / RoboTwin-CL
Descriptor horizon	10 steps	8 steps	50 steps
Candidate boundary	Demonstration	Episode and valid video prefix	Episode and instruction segment
Action channels	7-dimensional actions	7-dimensional actions	16-dimensional absolute EEF actions
Additional states	Gripper and joint states	Gripper and 6-dimensional EEF-pose proxy	Two gripper-state channels; no joint-state channel
Anchor visual summary	Agent and wrist views	None	Head and left-wrist views

Table 8: **Anchor configurations across policies and benchmarks.** Target proportions are measured in chunks relative to the per-task replay quota of $b = 1{,}000$. Realized proportions may exceed the upper target because demonstrations are retained in full; the total replay quota remains fixed.

Policy	Benchmark	Target proportion	Complete anchors/task
BCTransformer	LIBERO-Spatial	10–20%	1–10
BCTransformer	LIBERO-Object	25–35%	1–10
BCTransformer	LIBERO-Goal	25–35%	1–10
BCTransformer	LIBERO-Long	Length-dependent	Exactly 1
Qwen-GR00T	LIBERO-Spatial	25–35%	1–10
Qwen-GR00T	LIBERO-Object	25–35%	1–10
Qwen-GR00T	LIBERO-Goal	25–35%	1–10
Qwen-GR00T	LIBERO-Long	25–35%	1–10
Qwen-OFT	RoboTwin-CL (8 tasks)	25–35%	1–10

within that segment and repeat its final action when fewer than 50 future actions remain. Video-tail truncation is applied jointly to the modalities before constructing these segment indices. Where visual summaries are enabled, the anchor descriptor uses eight uniformly sampled frames per view and a 4×4 RGB grid. The two VLA adapters share the discovery, anchor-gating, coverage, and completion hyperparameters specified in Appendix A; their input mappings and horizons differ as shown in the table.

The settings in Table 8 are shared across seeds within each policy–benchmark pair. For BCTransformer on LIBERO-Long, we retain one complete demonstration per task; its length determines the realized anchor proportion. All other reported settings permit one to ten complete anchors per task while targeting the stated chunk proportions. The anchor-proportion ablation overrides these targets with 0%, 10–20%, 25–35%, or 45–55%; the 0% setting disables anchors.

B.5 Ablation Configurations

All ablations use LIBERO-Goal and LIBERO-Spatial with BCTransformer, a quota of 1,000 chunks per task, and training seeds 10000 and 10001. Table 9 gives the implemented target intervals. Their upper bounds are soft because selected anchor demonstrations remain complete; actual retained fractions can differ from these intervals. A0 disables complete-demonstration anchors.

The component comparisons use A25 on Goal and A15 on Spatial. The full-method configurations in Figures 5 and 6 use the matching two-seed results, rather than the three-seed aggregate in the main comparison. The no-coverage variant sets the explicit coverage quota to zero and transfers the released capacity to completion, while retaining the anchor-selection procedure and structural description.

Table 9: **Anchor-proportion configurations for the ablation study.** A0 disables complete-demonstration anchors. Component ablations use A25 on LIBERO-Goal and A15 on LIBERO-Spatial.

Configuration	Target anchor proportion	Minimum anchor demonstrations
A0	0%	0
A15	10–20%	1
A25	25–35%	1
A50	45–55%	1

The completion controls protect all anchor and coverage indices. Uniform selects evenly spaced entries from the remaining candidates sorted by dataset index. Random samples the same number of candidates uniformly without replacement. Neither control uses the original behavioral-group quotas or quality scores, and both exactly fill the residual capacity. These controls modify only completion after the protected scaffold.

B.6 Metric Definitions and Reporting

Let $R_{j,i}$ denote the success rate on task i after learning stage j, for $j \geq i$, and let K be the number of tasks in the stream. We use $K = 10$ for each LIBERO suite and $K = 8$ for RoboTwin-CL. Success rates are expressed as fractions in the definitions below. The BCTransformer experiments use within-task learning curves as well as stage-end evaluations; the Qwen-GR00T and RoboTwin-CL comparisons use stage-end evaluations.

BCTransformer acquisition and lifecycle AUC. Let $p_{i,e}$ be the success rate on task i at evaluation index e during its own training. The $E = 11$ evaluations correspond to epochs $0, 5, \ldots, 50$. Let $e_i^\star$ be the first index attaining the maximum success rate on this curve. The corresponding checkpoint is retained for stage-end evaluation and initializes the next task. Following the implemented peak-holding convention, define

$$\widetilde{p}_{i,e} = \begin{cases} p_{i,e}, & e < e_i^\star, \\ p_{i,e_i^\star}, & e \geq e_i^\star, \end{cases} \qquad f_i = \frac{1}{E} \sum_{e=1}^{E} \widetilde{p}_{i,e}, \qquad E = 11. \tag{24}$$

This convention holds the first global peak at subsequent evaluation points while preserving the original curve before that peak. Each BCTransformer success rate is estimated from 20 rollouts with a horizon of at most 600 environment steps.

The forward-transfer score and lifecycle AUC are

$$\mathrm{FWT} = \frac{1}{K} \sum_{i=1}^{K} f_i, \tag{25}$$

$$\mathrm{AUC}_{\mathrm{BC}} = \frac{1}{K} \sum_{i=1}^{K} \frac{f_i + \sum_{j=i+1}^{K} R_{j,i}}{K - i + 1}. \tag{26}$$

FWT summarizes acquisition using current-task demonstrations; it is not a zero-shot transfer measure. Lifecycle AUC combines each task's within-task acquisition score with its later retention scores before averaging across tasks. Tables for the BCTransformer experiments abbreviate $\mathrm{AUC}_{\mathrm{BC}}$ as AUC.

Forgetting. All comparisons use the same negative-backward-transfer convention:

$$\mathrm{NBT} = \frac{1}{K} \sum_{i=1}^{K-1} \frac{1}{K - i} \sum_{j=i+1}^{K} \big(R_{i,i} - R_{j,i} \big). \tag{27}$$

For BCTransformer, the initial reference is the selected checkpoint's training-time success, $R_{i,i} = \max_e p_{i,e}$; later entries $R_{j,i}$, with $j > i$, are evaluations of task i after learning stage j. This is distinct from an independent reevaluation of the initial checkpoint. The task-wise diagnostic curves

in Section C.3 preserve their recorded stage-end measurements and do not replace this reference in the reported NBT. For each task, NBT averages its performance decline over all subsequent stages, then averages over all K tasks. The final task contributes zero because it has no subsequent stage. Improvements are not clipped: lower values are better, and negative values indicate positive backward transfer. This measure uses all later stages rather than only the final policy. Our normalization includes the final task's zero contribution; CLARE's NBT instead normalizes over the first $K-1$ tasks. Our adoption of CLARE below concerns AUC and does not change Equation 27.

Stage-end AUC and final success. For Qwen-GR00T, $R_{j,i}$ is measured using the final checkpoint of training stage j. For both the Qwen-GR00T and RoboTwin-CL comparisons, we compute AUC from stage-end success rates following CLARE (Römer et al., 2026):

$$\mathrm{AUC}_{\mathrm{stage}} = \frac{1}{K}\sum_{i=1}^{K}\frac{1}{K-i+1}\sum_{j=i}^{K}R_{j,i}. \tag{28}$$

This averages each task's success over its own and subsequent stages, then averages across tasks. Its acquisition term is $R_{i,i}$, whereas Equation 26 uses f_i. It therefore requires no within-task learning curve and is distinct from averaging the mean seen-task success across stages. Both panels of Figure 3 abbreviate $\mathrm{AUC}_{\mathrm{stage}}$ as AUC; neither reports FWT.

Final average success is

$$\mathrm{ASR} = \frac{1}{K}\sum_{i=1}^{K}R_{K,i}. \tag{29}$$

ASR summarizes the final policy across all tasks. NBT in these comparisons is computed with Equation 27. Empty retention sums are zero in all definitions above.

Reporting across seeds. We compute each metric independently for each complete task stream, then report the arithmetic mean and sample standard deviation across training seeds. The main BCTransformer comparison uses seeds 10000, 10001, and 10002. The reported Qwen-GR00T comparison uses the two completed seeds 42 and 10000, with evaluation seed fixed to 42. The ablations use seeds 10000 and 10001. The RoboTwin-CL experiments use two training seeds, 42 and 10000, for both methods. A percentage presentation multiplies both means and standard deviations by 100. Incomplete task streams are excluded from aggregate results, and any partial seed coverage is identified explicitly.

C Supplementary Results and Analysis

C.1 Task Scene Visualization

We show all 40 LIBERO tasks and the eight RoboTwin-CL tasks. Each panel uses the temporal midpoint of a recorded demonstration from the fixed external camera: `agentview_rgb` for LIBERO and `cam_high` (the head camera) for RoboTwin-CL. Task IDs follow training order, and labels abbreviate task instructions. LIBERO frames are displayed upright by reversing their stored vertical orientation; the scene content and framing are preserved.

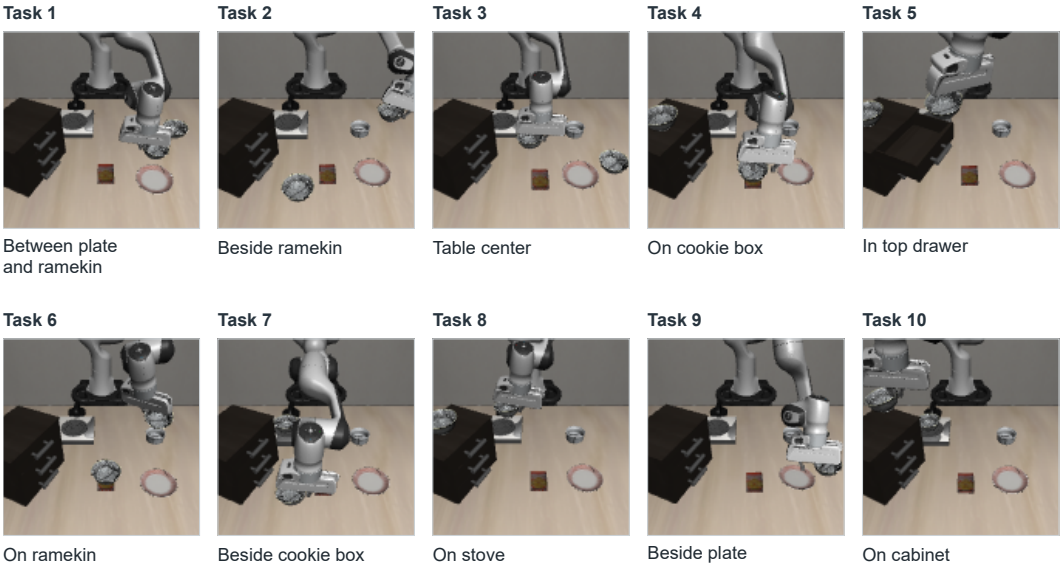

Figure 7: **Task scenes in LIBERO-Spatial.** All ten tasks place a black bowl on a plate. Labels identify the bowl's initial location; images show mid-demonstration scenes.



Figure 8: **Task scenes in LIBERO-Object.** Each task places a different object in the basket. Labels identify the manipulated objects in the ten tasks.

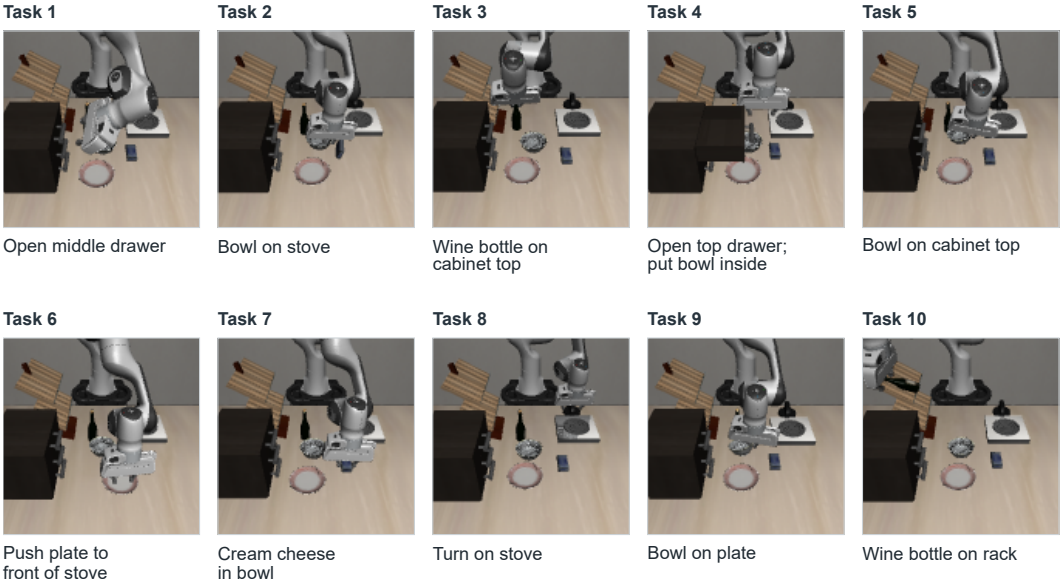

Figure 9: **Task scenes in LIBERO-Goal.** Mid-demonstration views illustrate the ten manipulation goals, including object placement, drawer interaction, and stove control.

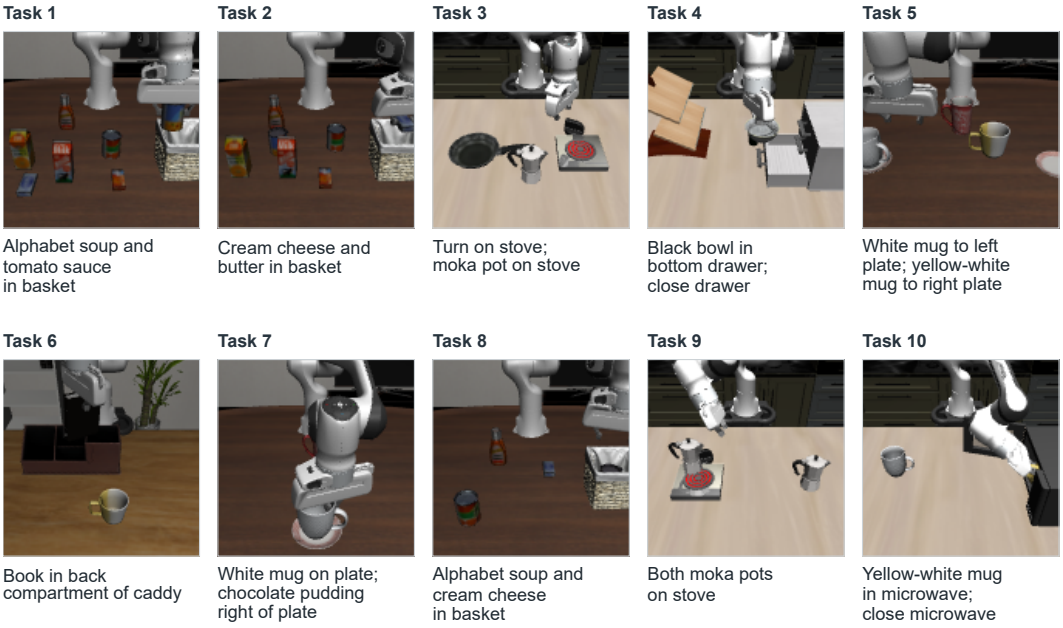

Figure 10: **Task scenes in LIBERO-Long.** The ten tasks include multi-object placement and multi-stage manipulation. Labels summarize the complete task instructions, while each image shows one middle frame.

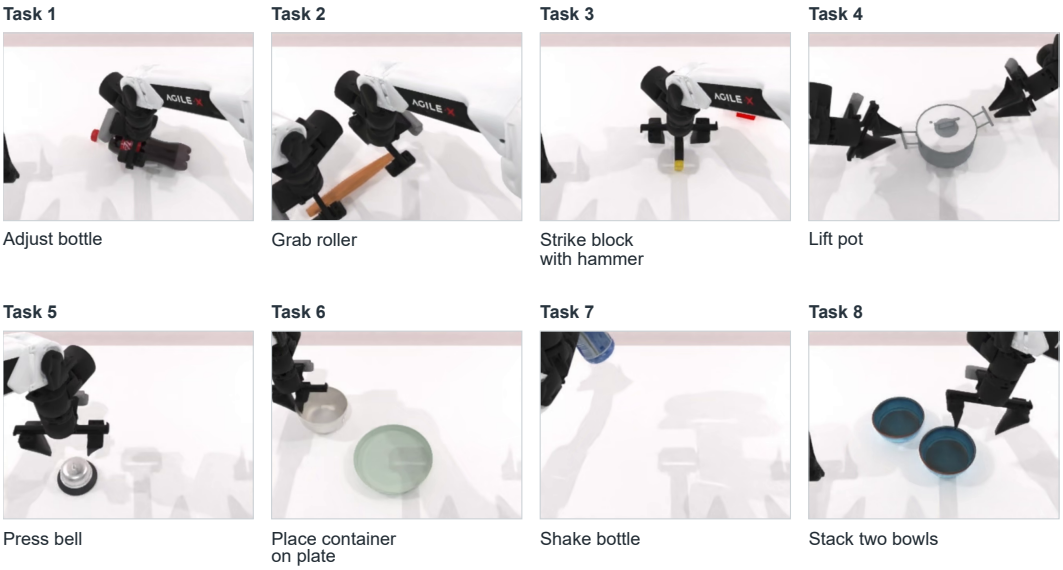

Figure 11: **Task scenes in RoboTwin-CL.** The eight tasks are shown in training order using the fixed head-camera view. Each image is taken at its demonstration's temporal midpoint. The Shake bottle panel uses episode 29, with the bottle partially outside the original view; the other seven tasks use episode 0.

C.2 Full Numerical Results

Tables 10 and 11 provide the numerical results underlying Figures 3, 4, 5, and 6. All entries report means and sample standard deviations over two seeds. The ablations use the matched two-seed full-method configurations, not the three-seed aggregates in Table 1. Metric definitions follow Section B.6.

Table 10: **Numerical results across policy architectures and task domains.** Qwen-GR00T is evaluated on the four LIBERO suites and Qwen-OFT on RoboTwin-CL, using the stage-end AUC protocol. Results are mean ± sample standard deviation over two seeds, matching Figure 3. Bold indicates the better mean for each policy–suite pair; higher ASR and AUC and lower NBT are better.

Task suite	Method	ASR ↑	NBT ↓	AUC ↑
Qwen-GR00T on LIBERO				
Spatial	ER	84.40 ± 1.70	8.57 ± 0.13	87.97 ± 2.47
	HMR	**91.40** ± 0.57	**1.94** ± 0.42	**92.25** ± 1.16
Object	ER	96.30 ± 1.56	2.60 ± 0.18	95.02 ± 0.47
	HMR	**96.90** ± 0.42	**1.17** ± 0.86	**97.38** ± 0.83
Goal	ER	87.90 ± 1.84	9.00 ± 1.14	87.48 ± 1.28
	HMR	**93.90** ± 1.84	**2.11** ± 0.07	**93.58** ± 1.55
Long	ER	67.60 ± 1.41	19.15 ± 0.72	73.84 ± 0.59
	HMR	**76.70** ± 0.99	**8.34** ± 3.52	**78.26** ± 0.30
Qwen-OFT on RoboTwin-CL				
RoboTwin-CL	ER	34.75 ± 0.35	4.30 ± 1.18	35.28 ± 1.56
	HMR	**40.25** ± 2.47	−**1.16** ± 1.28	**38.12** ± 2.23

Table 11: **Numerical results for the memory-construction ablations.** BCTransformer is evaluated on Goal and Spatial with $b = 1{,}000$ chunks per task. Values are mean ± sample standard deviation over two matched seeds. In (a), intervals denote target anchor proportions rather than exact realized fractions. In (b), anchors remain fixed and removing coverage releases its capacity to completion. In (c), all strategies share the same anchors and coverage chunks. Parts (b) and (c) use A25 on Goal and A15 on Spatial. Bold indicates the best mean within each part and suite.

	LIBERO-Goal			LIBERO-Spatial		
Configuration	FWT ↑	NBT ↓	AUC ↑	FWT ↑	NBT ↓	AUC ↑
(a) Target anchor proportion						
A0: 0%	69.09 ± 0.00	**18.97** ± 1.80	66.26 ± 2.76	68.00 ± 1.67	21.33 ± 4.75	63.43 ± 1.82
A15: 10–20%	65.05 ± 11.38	20.07 ± 9.57	61.33 ± 3.11	**70.61** ± 1.06	**19.83** ± 1.05	**67.14** ± 0.71
A25: 25–35%	**69.87** ± 2.57	19.61 ± 6.25	**66.82** ± 2.07	68.53 ± 0.16	19.87 ± 4.28	64.14 ± 4.16
A50: 45–55%	62.93 ± 2.22	22.02 ± 0.56	58.55 ± 1.56	65.89 ± 1.19	22.29 ± 3.01	60.40 ± 2.67
(b) Structural coverage						
Without coverage	66.30 ± 6.58	21.22 ± 2.88	63.43 ± 3.17	66.80 ± 2.47	**17.67** ± 1.15	65.95 ± 5.25
With coverage	**69.87** ± 2.57	**19.61** ± 6.25	**66.82** ± 2.07	**70.61** ± 1.06	19.83 ± 1.05	**67.14** ± 0.71
(c) Residual completion						
Random	64.57 ± 6.46	21.92 ± 4.53	60.57 ± 3.17	64.57 ± 5.49	**16.94** ± 1.12	64.27 ± 2.52
Uniform	66.07 ± 3.12	**17.11** ± 2.16	64.89 ± 5.64	68.75 ± 1.77	17.53 ± 0.41	**68.24** ± 0.30
Representative	**69.87** ± 2.57	19.61 ± 6.25	**66.82** ± 2.07	**70.61** ± 1.06	19.83 ± 1.05	67.14 ± 0.71

C.3 Task-wise Acquisition and Retention

Each panel follows one task in its stream's training order. Markers show seed means and shading denotes one sample standard deviation, clipped to 0–100%. All tasks are included, without smoothing or averaging across tasks.

C.3.1 BCTransformer: Stage-End Performance

Curves preserve recorded stage-end evaluations from each task's acquisition onward, using three seeds. Where available, the initial point is an independent reevaluation, rather than the training-time peak used as the NBT reference in Section B.6. The final task has only one observation.



Figure 12: Task-wise success of BCTransformer on LIBERO-Spatial. Each panel tracks a fixed task through subsequent training stages. Curves show mean success and one sample standard deviation across three seeds.

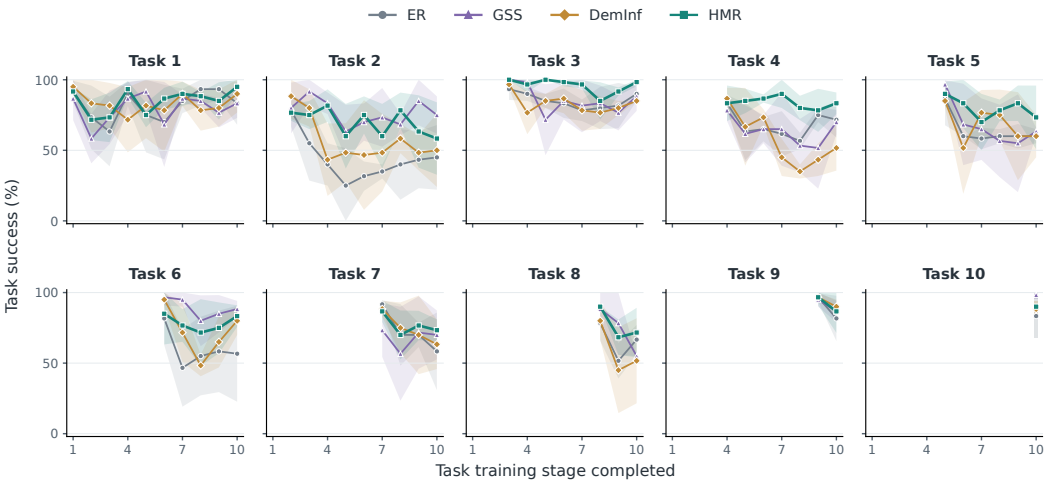

Figure 13: Task-wise success of BCTransformer on LIBERO-Object. Each panel tracks a fixed task through subsequent training stages. Curves show mean success and one sample standard deviation across three seeds.



Figure 14: Task-wise success of BCTransformer on LIBERO-Goal. Each panel tracks a fixed task through subsequent training stages. Curves show mean success and one sample standard deviation across three seeds.



Figure 15: Task-wise success of BCTransformer on LIBERO-Long. Each panel tracks a fixed task through subsequent training stages. Curves show mean success and one sample standard deviation across three seeds.

C.3.2 QWEN-GR00T: STAGE-END PERFORMANCE

These curves follow the same tasks after each completed training stage, using two training seeds. The available VLA records contain stage-end evaluations only, so curves begin after each task is first learned; no training-start or within-task observations are inferred.

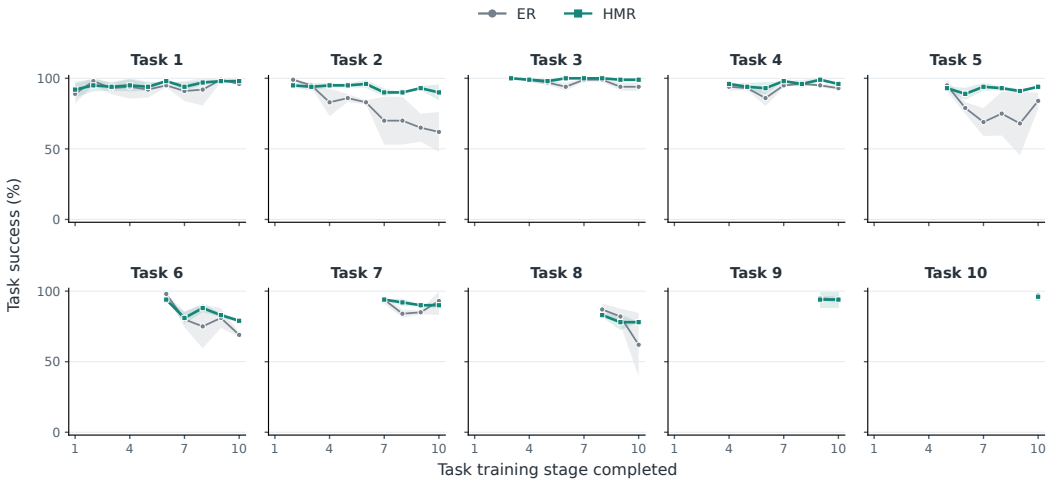

Figure 16: Task-wise success of Qwen-GR00T on LIBERO-Spatial. Curves begin at the first post-training evaluation of each task and show mean success and one sample standard deviation across two seeds.

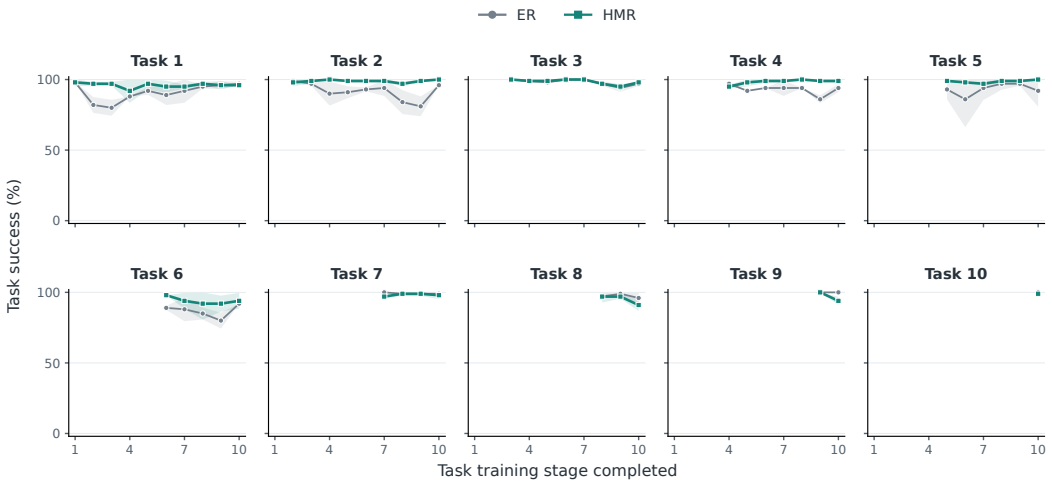

Figure 17: Task-wise success of Qwen-GR00T on LIBERO-Object. Curves begin at the first post-training evaluation of each task and show mean success and one sample standard deviation across two seeds.

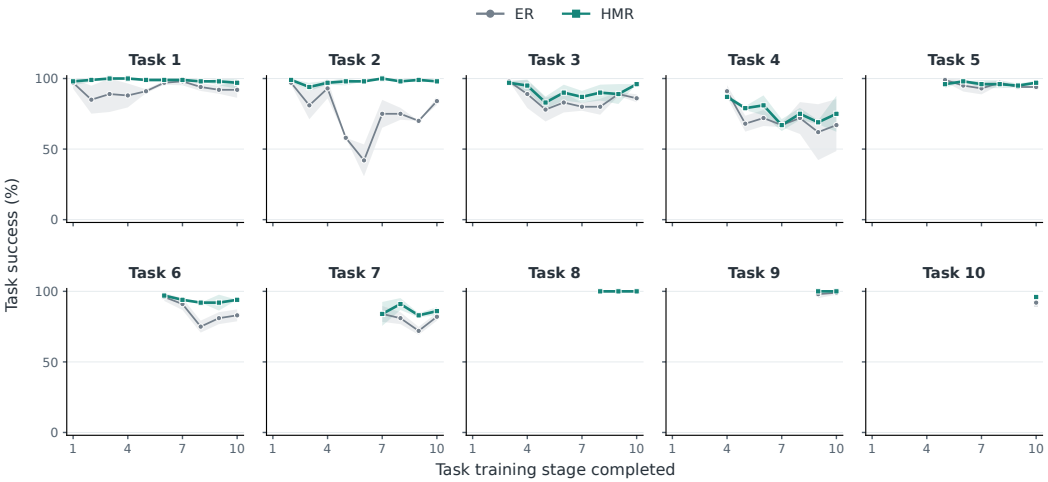

Figure 18: Task-wise success of Qwen-GR00T on LIBERO-Goal. Curves begin at the first post-training evaluation of each task and show mean success and one sample standard deviation across two seeds.

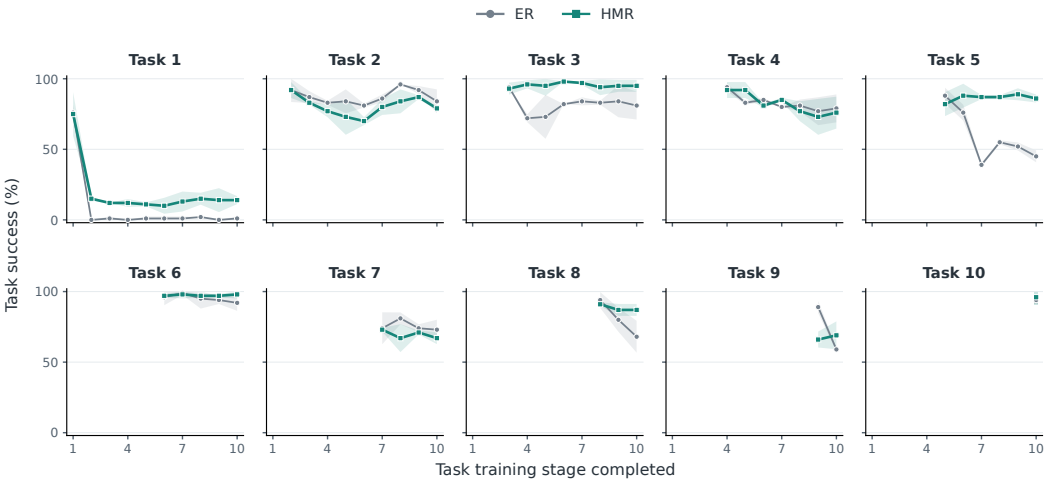

Figure 19: Task-wise success of Qwen-GR00T on LIBERO-Long. Curves begin at the first post-training evaluation of each task and show mean success and one sample standard deviation across two seeds.

C.3.3 QWEN-OFT: STAGE-END PERFORMANCE

We show all eight tasks in the RoboTwin-CL stream, using the same stage-end convention and two training seeds. Following individual tasks makes task-specific gains and losses visible alongside the aggregate results.

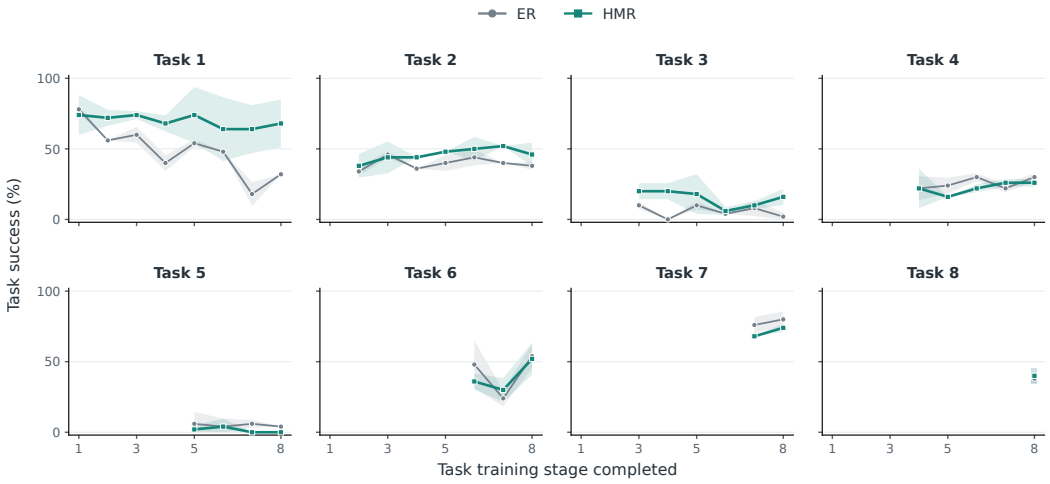

Figure 20: Task-wise success of Qwen-OFT on RoboTwin-CL. Each panel follows one of the eight tasks from its first post-training evaluation to the end of the stream. Shading denotes one sample standard deviation across two seeds.

C.3.4 BCTRANSFORMER: WITHIN-TASK ACQUISITION

These plots show raw current-task success at epochs $0, 5, \ldots, 50$, using three seeds. Epoch 0 precedes updates on the current task, and the observations are not peak-held. The checkpoint retained for subsequent stages may precede epoch 50, and some stage-end measurements reevaluate that checkpoint. Thus, the final training observation and the first stage-end observation need not coincide.



Figure 21: Within-task acquisition of BCTransformer on LIBERO-Spatial. Each panel shows raw success while learning its current task, from epoch 0 through epoch 50. Curves show the mean and one sample standard deviation across three seeds.

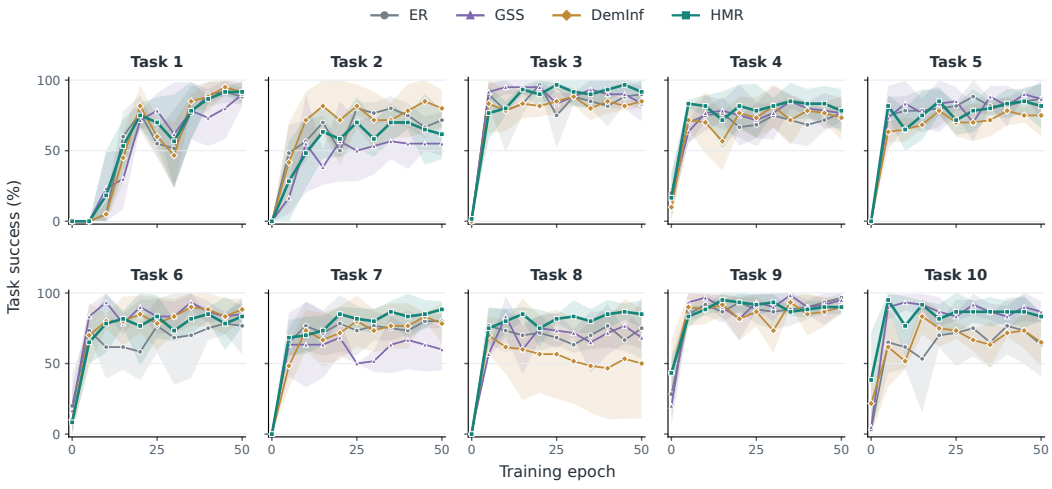

Figure 22: Within-task acquisition of BCTransformer on LIBERO-Object. Each panel shows raw success while learning its current task, from epoch 0 through epoch 50. Curves show the mean and one sample standard deviation across three seeds.

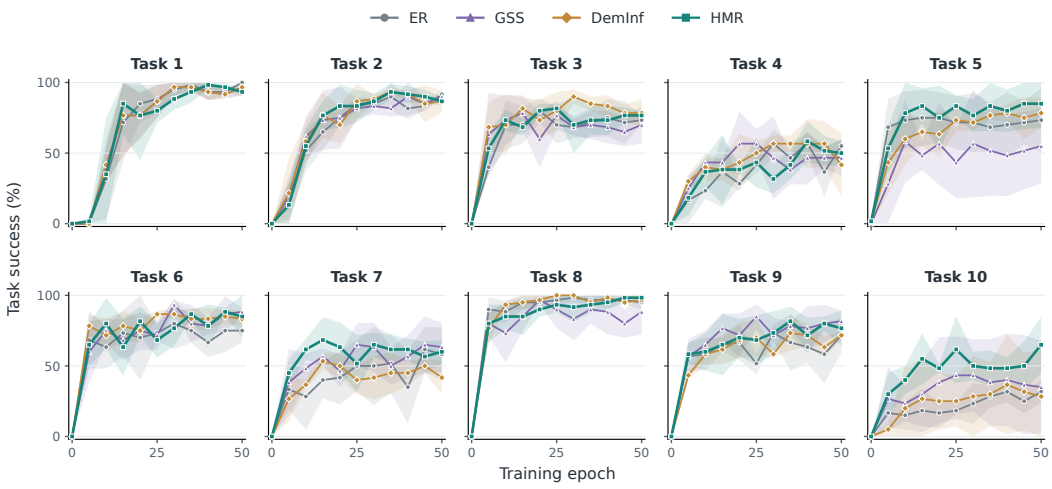

Figure 23: Within-task acquisition of BCTransformer on LIBERO-Goal. Each panel shows raw success while learning its current task, from epoch 0 through epoch 50. Curves show the mean and one sample standard deviation across three seeds.



Figure 24: Within-task acquisition of BCTransformer on LIBERO-Long. Each panel shows raw success while learning its current task, from epoch 0 through epoch 50. Curves show the mean and one sample standard deviation across three seeds.