LEARNING NEW TASKS VIA REUSABLE SKILLS: SKILL-COMPOSITIONAL EXPERTS FOR EMBODIED CONTINUAL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Embodied Continual Learning (ECL) aims to enable robots to continually acquire new manipulation tasks while retaining previously learned behaviors under closed-loop control. In ECL, feature drift can propagate through sequential decision-making under closed-loop control, turning representation changes into compounding behavioral deviations on previously learned tasks. A key challenge in ECL lies in structured skill reuse across continually evolving tasks, since existing methods primarily focus on skill learning without explicitly organizing them for coherent task execution. To address this issue, we propose SCE, a Skill-Compositional Experts framework for ECL. SCE builds a skill base via Compositional Skill Grounding (CSG), which decomposes task demonstrations into reusable skills. Based on this, Dual Execution-and-Transition Experts (DETE) enable new task learning through skill composition, where one branch ensures skill execution and the other supports transitions between skills for coherent behavior. Experiments on LIBERO benchmarks and real-world manipulation tasks show that SCE improves retention and overall task performance. Further feature drift analyses and ablation studies verify the effectiveness of our method. The code and real-world datasets will be released upon acceptance of the paper.

1 INTRODUCTION

Continual learning (CL) is essential for building systems that can adapt to evolving tasks over time (Parisi et al., 2019; De Lange et al., 2022). Despite substantial progress, most existing CL methods are studied in settings that lack closed-loop interaction with dynamic environments, where data does not evolve through robot-environment interaction. In contrast, embodied robots learn through continuous interaction with the environment, where observations are coupled with actions and evolve over time, giving rise to Embodied Continual Learning (ECL) (Lesort et al., 2020; Jia et al., 2025). In this setting, robots must continually acquire new embodied tasks while preserving the ability to reliably execute previously learned behaviors under dynamically changing states.

In such interactive environments, catastrophic forgetting has direct behavioral consequences. As illustrated in Fig. 1(a), the feature representations of Vision-Language-Action (VLA) models can progressively drift toward new tasks. Under closed-loop control, this drift changes subsequent observations and states, so deviations can compound through sequential decision-making and degrade previously learned behaviors (Ross et al., 2011; Spencer et al., 2021). Therefore, ECL is crucial for improving the generalization and long-term adaptability of embodied robots, which are essential for real-world deployment (Black et al., 2024; Kim et al., 2025; Zitkovich et al., 2023).

Existing ECL methods primarily mitigate catastrophic forgetting at the task or skill level. Task-level approaches preserve old knowledge through replay of previous task demonstrations (Gao et al., 2021) or task-specific expert routing mechanisms (Jia et al., 2025). However, they treat each task as an isolated learning unit, limiting the exploitation of shared patterns across tasks. This is suboptimal for embodied manipulation, where skill-level knowledge is essential for generalizing across diverse tasks. To address this limitation, skill-level methods (Lee et al., 2024) shift the focus to learning reusable skill primitives across tasks. Although effective, existing methods mainly focus on skill discovery (Wan et al., 2024) or evolution (Wu et al., 2025), without explicitly modeling their organization for new

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.

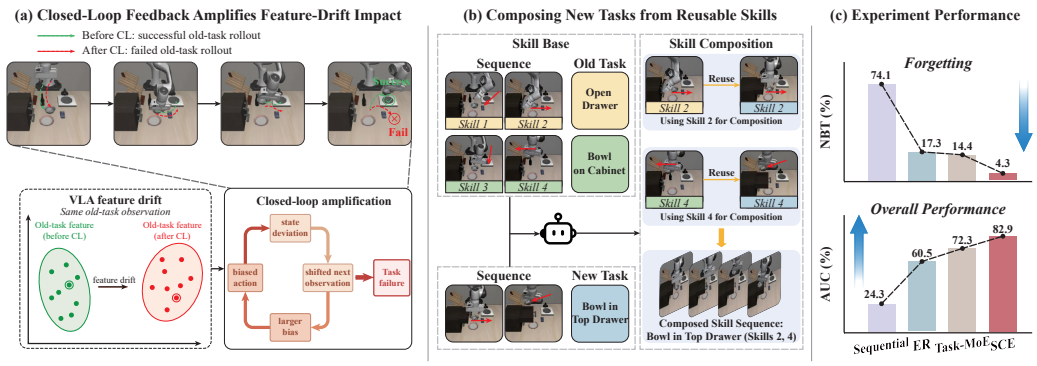


Figure 1: (a) Closed-loop control amplifies the impact of old-task feature drift. (b) New tasks can be composed from reusable skills in the skill base. (c) Forgetting and overall performance comparison on LIBERO-Goal.

task execution. As a result, this lack of structured skill organization leads to inconsistent skill reuse under sequential tasks, amplifying feature drift under closed-loop control. This highlights the need for structured skill reuse in continual learning.

Motivated by this insight, we propose a compositional view of ECL, where each task can be represented as a composition of reusable skills. Based on this view, we introduce a Skill-Compositional Experts (SCE) framework for ECL. As illustrated in Fig. 1(b), SCE maintains a skill base that stores reusable skills acquired from previous tasks, allowing new tasks to be learned through composition of relevant skills. Specifically, SCE consists of two components. First, Compositional Skill Grounding (CSG) decomposes task demonstrations into reusable skills and organizes them into the skill base, enabling reuse of skill knowledge across tasks. Second, the Dual Execution-and-Transition Experts (DETE) module further augments the VLA action decoder with two complementary expert branches. The Execution Expert Branch models skill-specific action patterns for the execution of reusable skills, while the Transition Expert Branch models cross-skill transition patterns for skill switching during task execution. In addition, DETE employs the Adaptive Fusion Module to dynamically combine the outputs of the two branches for closed-loop action generation. Through this design, SCE maintains stable skill execution while flexibly handling skill transitions, thereby reducing feature drift during ECL. Comprehensive experiments on both simulated and real-world datasets demonstrate the advantages of SCE. In simulation, SCE achieves lower forgetting while maintaining stronger overall performance, as shown in Fig. 1(c). Real-world robot experiments further verify its robustness in physical manipulation scenarios. In-depth analyses further show that SCE effectively alleviates feature drift under closed-loop control. Our contributions are summarized as follows:

(1) We propose SCE, a Skill-Compositional Experts framework for ECL that enables continual task learning through composition of reusable skills, while preserving previously learned behaviors.

(2) We introduce CSG and DETE. CSG decomposes task demonstrations into reusable skills and organizes them into a skill base, while DETE supports skill execution and cross-skill transitions for coherent skill composition.

(3) Extensive experiments in both simulated and real-world manipulation environments demonstrate the effectiveness of SCE.

2 Related Work

2.1 Continual Learning

Methods for addressing catastrophic forgetting in Continual Learning (CL) are commonly grouped into four categories: Regularization-based (Li & Hoiem, 2018; Tao et al., 2020b;a; Wu et al., 2021; Caccia et al., 2020) approaches constrain parameter or feature updates to preserve old knowledge. Architecture-based approaches (Yan et al., 2021; Mallya & Lazebnik, 2018; Liu et al., 2021; Zhao et al., 2024) expand, prune, or reallocate model capacity to reduce interference among tasks.

Rehearsal-based approaches (Rolnick et al., 2019; Castro et al., 2018; Wu et al., 2019; Hu et al., 2022b; 2021; Lin et al., 2023) store representative samples from previous tasks and replay them during current learning stages. More recently, pre-trained model-based approaches (Wang et al., 2022b;a; Smith et al., 2023; Jung et al., 2023; Tang et al., 2023; Zhou et al., 2024; Li et al., 2025; Peng et al., 2025) adapt strong backbones with prompts, adapters, or other parameter-efficient tuning strategies to improve the stability-plasticity trade-off. Although these methods have achieved strong progress in mitigating catastrophic forgetting, they are mostly developed for settings where the evaluation inputs are fixed and independent of the model's predictions. This assumption differs from Embodied Continual Learning (ECL), where a robot performs tasks under closed-loop control, such that each predicted action influences subsequent observations and states. As a result, forgetting in ECL goes beyond performance degradation on static samples: errors can propagate over sequential action execution and compound into degraded embodied behaviors. This makes ECL inherently more challenging than conventional CL and highlights the need for continual learning mechanisms tailored to closed-loop interaction.

2.2 Embodied continual learning

Embodied Continual Learning (ECL) extends CL to embodied robots that acquire sequential tasks through closed-loop interaction with the environment. Recent VLA models and generalist policies have achieved strong progress in embodied manipulation (Zitkovich et al., 2023; Kim et al., 2025; Black et al., 2024; Bu et al., 2025; Reuss et al., 2025). Nevertheless, these models remain susceptible to catastrophic forgetting when applied to ECL settings. Existing ECL methods primarily mitigate this problem at either the task level or the skill level. Task-level methods (Liu et al., 2024; Römer et al., 2026; Song et al., 2025) treat each task or task cluster as the basic unit for organizing continual knowledge. For example, CRIL (Gao et al., 2021) preserves prior task knowledge by reconstructing previous task trajectories with generative models. MoILE (Jia et al., 2025) instead organizes knowledge through task clusters and routes inputs to hierarchical LoRA (Hu et al., 2022a) experts, reducing interference via task-aware expert selection. M2Distill (Roy et al., 2025) preserves prior knowledge through multimodal distillation for lifelong imitation learning. These methods demonstrate the effectiveness of task-level organization for reducing interference and preserving learned behaviors. Nevertheless, task-level organization of continual knowledge may obscure shared behavioral structures across tasks. Skill-level methods organize continual knowledge around finer-grained skill-oriented structures, enabling reusable behaviors to be shared across tasks (Yao et al., 2025; Lee et al., 2025; Zheng et al., 2025; Xu & Nie, 2026). For example, IsCiL (Lee et al., 2024) focuses on retrievable skills, where prototype-indexed skill adapters are retrieved for efficient continual task adaptation. LOTUS (Wan et al., 2024) targets skill discovery, identifying temporal segments from demonstrations and organizing them into an expanding skill-policy library for manipulation. Stellar VLA (Wu et al., 2025) promotes continually evolving skill knowledge, maintaining a task-skill knowledge space to guide VLA-based ECL. T2S (Zhang et al., 2025a) scales reusable knowledge through language-selected parameter tokens. These methods show that organizing continual knowledge at the skill level can improve knowledge sharing and retention across tasks. SCE builds on this skill-level view by explicitly modeling skill-specific execution and cross-skill transitions to compose reusable skills for new tasks.

3 Method

As shown in Fig. 2, SCE enables skill-compositional embodied continual learning by constructing a skill base and composing skills across tasks. It consists of CSG (Compositional Skill Grounding) and DETE (Dual Execution-and-Transition Experts). Specifically, CSG decomposes task demonstrations into reusable skills and organizes them into the skill base. DETE augments the VLA action decoder with two complementary expert branches that leverage reusable skills to model skill-specific behavioral patterns and cross-skill compositional knowledge, supporting execution and coherent skill composition across tasks.

3.1 Problem definition

In embodied continual learning, a VLA model is required to sequentially acquire new manipulation tasks while retaining previously learned ones. Let $\mathcal{D} = \{\mathcal{D}^1, \mathcal{D}^2, \ldots, \mathcal{D}^T\}$ denote a stream of T

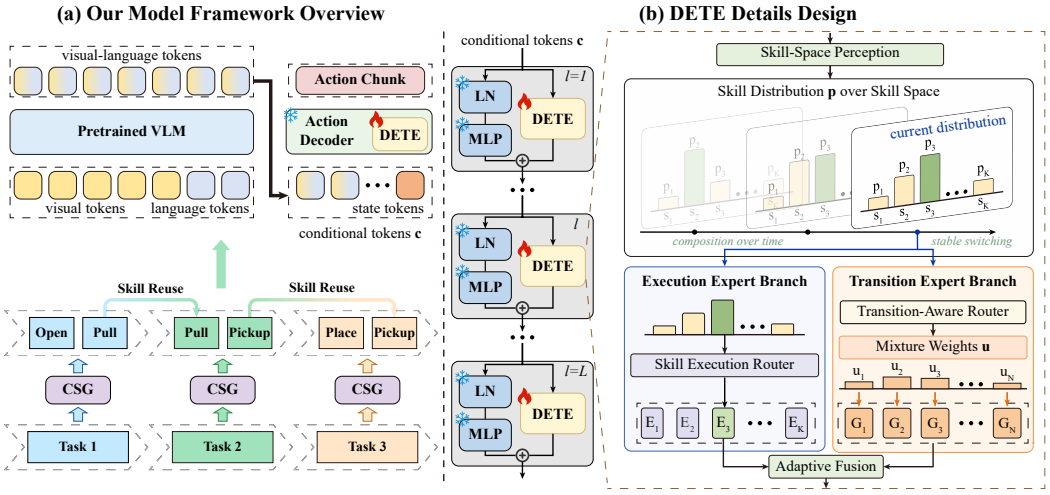


Figure 2: **Overall framework of the proposed method.** (a) The framework processes a sequential task stream by decomposing task demonstrations into reusable skills. DETE is integrated into the pretrained VLA action decoder to support skill-compositional embodied continual learning. (b) DETE combines skill-space perception with Execution and Transition Expert branches for skill-specific behavior modeling and coherent skill composition.

stages. At the t-th stage, the current dataset $\mathcal{D}^t$ and a bounded replay memory from previous stages are accessible. Each sample in $\mathcal{D}^t$ is represented as $x_i = (o_i, l_i, r_i, s_i, \mathbf{A}_i)$, where o_i denotes the visual observation, l_i is the language instruction, r_i represents the robot state, s_i denotes the associated skill annotation constructed as described in Sec. 3.3, and $\mathbf{A}_i = (\mathbf{a}_i, \mathbf{a}_{i+1}, \ldots, \mathbf{a}_{i+H-1})$ denotes the corresponding H-step action chunk, with $\mathbf{a}_{i+h}$ denoting the robot action at future offset h for $h = 0, \ldots, H-1$. Let $\mathcal{T}^t$ denote the set of tasks introduced at stage t, where $\forall i \neq j$, $\mathcal{T}^i \cap \mathcal{T}^j = \varnothing$. The model is sequentially trained on $\mathcal{D}^1, \mathcal{D}^2, \ldots, \mathcal{D}^T$, and after training on $\mathcal{D}^t$, it is evaluated on all encountered tasks $\mathcal{T}^{1:t} = \bigcup_{k=1}^{t} \mathcal{T}^k$.

3.2 Overall Architecture

Fig. 2(a) illustrates the overall architecture. Our method builds on a pretrained VLA model and adds the DETE module to the action decoder. Given a sample $x = (o, l, r, s, \mathbf{A})$, the model predicts actions from (o, l, r). The skill annotation s provides auxiliary supervision for organizing the decoder around reusable skills. During ECL at stage t, DETE is optimized only with the samples available at that stage; skills that appear only in future stages provide no positive skill supervision or action-generation data before their corresponding stages are reached. We summarize the forward process as follows:

$$\begin{aligned} \mathbf{z} &= \Phi(o, l), \qquad \mathbf{e} = \Psi(r), \\ \mathbf{c} &= [\mathbf{z}; \mathbf{e}], \qquad \hat{\mathbf{A}} = \Theta(\mathbf{c}), \end{aligned} \tag{1}$$

where Φ denotes the pretrained vision-language encoder, $\mathbf{z}$ is the visual-language token, Ψ denotes the robot-state projection layer, $\mathbf{e}$ is the state token, and $\mathbf{c}$ is the conditional token formed by concatenating $\mathbf{z}$ and $\mathbf{e}$. The action decoder Θ predicts the corresponding H-step continuous action chunk $\hat{\mathbf{A}} = (\hat{\mathbf{a}}_0, \hat{\mathbf{a}}_1, \ldots, \hat{\mathbf{a}}_{H-1})$, where $\hat{\mathbf{a}}_j$ denotes the predicted action at future offset j.

3.3 Compositional Skill Grounding

This section details how CSG performs task decomposition over demonstrations to obtain reusable skills. CSG proceeds in two phases: Phase 1 partitions each demonstration into temporally coherent segments according to robot-state dynamics, and Phase 2 grounds these segments with a Vision-Language Model (VLM) to induce reusable skills and update a skill base $\mathcal{B}$. The skill base is maintained across demonstrations and initialized as $\mathcal{B} = \emptyset$ before processing the demonstration stream.

Phase 1: State-Aware Task Decomposition. Let $\mathcal{X} = (\mathcal{O}, \mathcal{R}, l)$ denote a raw task demonstration of temporal length F, where $\mathcal{O} = \{o_i\}_{i=1}^{F}$ is the observation sequence, $\mathcal{R} = \{r_i\}_{i=1}^{F}$ is the robot-state sequence, and l is the language instruction. Inspired by the keyframe intuition of Chain-of-Action (Zhang et al., 2025b), we define two state-based rules for temporal decomposition. Rule 1 marks a temporal boundary when the gripper state changes between adjacent timesteps. Rule 2 marks a temporal boundary when the end-effector motion trend changes abruptly. CSG applies these rules to the robot-state sequence $\mathcal{R}$ and obtains a set of temporal boundary indices. These indices split the demonstration into M contiguous temporal segments $\{\mathcal{X}^{(m)}\}_{m=1}^{M}$. Each segment is defined as $\mathcal{X}^{(m)} = (\mathcal{O}^{(m)}, \mathcal{R}^{(m)}, l)$, where $\mathcal{O}^{(m)}$ and $\mathcal{R}^{(m)}$ denote the corresponding observation and robot-state subsequences.

Phase 2: VLM-Grounded Skill Induction. After temporal decomposition, CSG processes the temporal segments $\{\mathcal{X}^{(m)}\}_{m=1}^{M}$ sequentially using the current skill base $\mathcal{B}$. For each temporal segment $\mathcal{X}^{(m)}$, the VLM (Bai et al., 2025) grounds the segment observation $\mathcal{O}^{(m)}$ and the language instruction l with reference to the current skill base $\mathcal{B}$, and proposes a candidate skill $\hat{s}^{(m)}$. The candidate $\hat{s}^{(m)}$ is then used to retrieve matched skills from the skill base $\mathcal{B}$. If no existing skill in $\mathcal{B}$ matches $\hat{s}^{(m)}$, $\hat{s}^{(m)}$ is added to $\mathcal{B}$ as a new reusable skill. Otherwise, the matched skill is selected from $\mathcal{B}$. The resulting skill assigned to $\mathcal{X}^{(m)}$ is denoted as $s^{(m)}$. After all temporal segments are processed, CSG obtains the updated skill base $\mathcal{B}$ together with the segment-level skill assignments $\{(\mathcal{X}^{(m)}, s^{(m)})\}_{m=1}^{M}$.

The assignments $\{(\mathcal{X}^{(m)}, s^{(m)})\}_{m=1}^{M}$ are then converted into the sample-level skill annotations used in Sec. 3.1. Let $\mathcal{S} = \{s_i\}_{i=1}^{F}$ denote the resulting annotation sequence for demonstration $\mathcal{X}$. For any sample indexed by i within temporal segment $\mathcal{X}^{(m)}$, we set $s_i = s^{(m)}$. This yields a skill-annotated demonstration, while the skill base $\mathcal{B}$ is updated across the demonstration stream. CSG is applied to both simulated and real-world task demonstrations. Appendix A details the algorithm, segmentation settings, and VLM prompt; Appendix B provides skill vocabularies and annotation examples.

3.4 Dual Execution-and-Transition Experts

DETE adopts a dual-expert design with two complementary branches to support embodied continual learning through skill composition. The Execution Expert Branch focuses on modeling skill-specific behaviors, ensuring execution of individual skills. The Transition Expert Branch captures common knowledge across different skills, facilitating transitions among skills during composition. Together, these branches enable the skill-compositional process by combining skill-specific execution specialization with stable cross-skill transition modeling.

We attach DETE in parallel to the pretrained action decoder, while the decoder remains responsible for transforming encoded visual-language and robot-state features into executable robot actions. Since skill composition must ultimately be reflected in temporally coherent action sequences, this parallel interface allows DETE to provide skill-specific execution patterns and cross-skill transition structure to the decoder during action decoding. For one decoder layer, let $\mathbf{y}$ denote the feature entering DETE; we omit the layer index for clarity. Each expert is implemented as a bottleneck adapter with independent parameters. Appendix C specifies the insertion points, adapter ranks, and trainable modules.

Skill-Space Perception. For skill composition along the action sequence, DETE needs a sample-level skill distribution that characterizes each input in terms of reusable skills and reflects changes in skill around transitions. At stage t, this distribution is defined on the K_t-dimensional skill space $\mathcal{V}_t$ constructed from the current skill base $\mathcal{B}_t$, where $K_t = |\mathcal{B}_t|$. When CSG discovers a new skill, SCE appends one classifier output, its aligned Execution Expert, and the corresponding input column of the transition projection while preserving all existing skill indices and parameters. The Transition Expert pool remains fixed. Full expansion details are provided in Appendix C.1.

DETE derives the skill distribution $\mathbf{p} \in \mathbb{R}^{K_t}$ by mapping the conditional token $\mathbf{c}$ to the current skill space $\mathcal{V}_t$ and normalizing the result with a softmax function. During training, this distribution is supervised by the automatically derived skill annotation:

$$\mathbf{p} = \mathrm{softmax}(\mathrm{MLP}(\mathbf{c})), \qquad \mathcal{L}_{\mathrm{skill}} = \mathrm{CE}(\mathbf{p}, \mathbf{s}). \tag{2}$$

Here, $\mathbf{s}$ denotes the one-hot target encoded from the automatically derived skill annotation s, and $\mathrm{CE}(\cdot,\cdot)$ denotes the cross-entropy loss. At inference time, DETE obtains $\mathbf{p}$ from the observation, instruction, and robot state without requiring ground-truth skill annotations.

Execution Expert Branch. During skill composition, each input is typically dominated by one skill that determines the local execution behavior. To stabilize skill-specific execution within a composed action sequence, DETE uses the skill distribution $\mathbf{p}$ to select the most relevant reusable skill. We instantiate the Execution Expert pool $\mathcal{E}_t = \{E_k\}_{k=1}^{K_t}$, where E_k is associated with the k-th dimension of the skill space $\mathcal{V}_t$.

The Skill Execution Router routes the input to the skill expert with the skill distribution $\mathbf{p}$:

$$\hat{k} = \arg\max_k p_k, \qquad \Delta\mathbf{y}^{\mathrm{exe}} = E_{\hat{k}}(\mathbf{y}). \tag{3}$$

Here, p_k denotes the k-th component of $\mathbf{p}$, measuring the relevance of the input to the k-th skill dimension in $\mathcal{V}_t$; $\hat{k}$ denotes the selected skill index and $\Delta\mathbf{y}^{\mathrm{exe}}$ denotes the skill-specific output of the selected expert. Under this design, execution-branch updates are confined to the expert associated with the currently involved skill, which reduces interference with unrelated skills and mitigates feature drift. By assigning one Execution Expert to each skill, this branch provides dedicated capacity for stable skill-specific execution during composition.

Transition Expert Branch. In real demonstrations, skill changes often extend over a temporal interval rather than occurring at a single instant. Around a skill transition, an input may contain cues from both the preceding skill and the subsequent skill. This ambiguity also appears at inference time, where ground-truth skill annotations are unavailable and the model must rely on the skill distribution $\mathbf{p}$. Therefore, the Transition Expert Branch treats $\mathbf{p}$ as a transition signal for modeling cross-skill dependencies during composition. We instantiate an N-expert Transition Expert pool $\mathcal{G} = \{G_j\}_{j=1}^{N}$ to model cross-skill transitions.

The Transition-aware Router maps the conditional token $\mathbf{c}$ into a routing-feature space and injects the skill distribution $\mathbf{p}$ to obtain a transition-aware routing feature $\mathbf{h}$:

$$\mathbf{h} = \mathrm{MLP}(\mathbf{c}) + W_{\mathrm{p}}\mathbf{p}. \tag{4}$$

Here, W_{p} is a linear projection that maps $\mathbf{p}$ to the dimension of $\mathbf{h}$. Based on $\mathbf{h}$, this router computes mixture weights $\mathbf{u} \in \mathbb{R}^N$ over $\mathcal{G}$ and aggregates their outputs $\Delta\mathbf{y}^{\mathrm{tr}}$:

$$\mathbf{u} = \mathrm{softmax}(W_{\mathrm{u}}\mathbf{h}), \qquad \Delta\mathbf{y}^{\mathrm{tr}} = \sum_{j=1}^{N} u_j G_j(\mathbf{y}), \tag{5}$$

where W_{u} is a linear projection and u_j denotes the j-th component of the mixture weights $\mathbf{u}$. Since different inputs may require different degrees of skill-specific execution and cross-skill transition modeling, DETE employs an Adaptive Fusion Module to compute an adaptive balancing coefficient from the transition-aware routing feature $\mathbf{h}$:

$$\alpha = \sigma(W_\alpha \mathbf{h}), \qquad \Delta\mathbf{y} = \alpha\Delta\mathbf{y}^{\mathrm{exe}} + (1-\alpha)\Delta\mathbf{y}^{\mathrm{tr}}. \tag{6}$$

Here, W_α is a linear projection, $\sigma(\cdot)$ denotes the sigmoid function, $\alpha \in [0,1]$ is the adaptive balancing coefficient, and $\Delta\mathbf{y}$ denotes the combined DETE output. The coefficient α adaptively balances skill-specific execution and cross-skill transitions through the transition-aware routing feature $\mathbf{h}$. Appendix E.7 analyzes how this balance changes around skill boundaries.

3.5 Optimization objective

The model is trained with the original action-generation objective and the auxiliary skill-distribution objective:

$$\mathcal{L} = \mathcal{L}_{\mathrm{flow}} + \lambda\mathcal{L}_{\mathrm{skill}}, \tag{7}$$

where $\mathcal{L}_{\mathrm{flow}}$ is the flow-matching loss of the pretrained VLA policy (Black et al., 2024), $\mathcal{L}_{\mathrm{skill}}$ is the skill supervision loss defined in Eq. 2, and the hyperparameter λ balances the two objectives. We use $\lambda = 0.2$; Appendix E.1 reports its sensitivity analysis.

Table 1: Main results under ECL. Results are mean ± standard deviation over three independent training seeds. Prior results (†) are taken from Stellar VLA (Wu et al., 2025).

Method	LIBERO-Goal				LIBERO-Long			
	FWT ↑	NBT ↓	AUC ↑	Final SR ↑	FWT ↑	NBT ↓	AUC ↑	Final SR ↑
UniVLA†	**87.5±0.1**	65.7±6.1	35.2±4.7	20.5±4.7	70.3±5.6	31.3±4.0	43.6±2.4	17.1±2.0
MoDE†	69.7±1.8	23.2±2.9	46.8±2.9	41.6±4.8	71.1±1.5	46.4±1.1	32.9±1.4	19.9±1.4
T-Stellar†	76.6±3.4	19.1±3.6	57.5±1.1	57.0±2.0	80.3±1.6	38.0±2.5	48.6±2.6	32.2±4.4
TS-Stellar†	71.8±4.0	25.9±2.7	47.5±2.5	47.7±1.4	**84.7±1.4**	40.8±0.6	51.4±1.4	39.7±1.6
IsCiL†	66.8±1.3	7.8±2.5	60.9±2.8	48.5±2.0	65.8±0.3	31.5±0.3	41.4±0.1	17.3±0.3
LOTUS†	46.8±4.3	37.7±3.9	18.5±1.5	10.8±1.1	32.3±2.3	2.8±2.7	29.4±0.5	31.8±2.3
Sequential	81.7±0.7	74.1±0.6	24.3±0.4	7.5±0.1	61.7±0.7	58.1±0.3	14.4±0.7	3.7±0.6
ER	74.7±0.4	17.3±0.3	60.5±0.3	44.9±0.1	73.3±1.4	22.5±0.4	55.2±1.2	50.9±0.9
Task-MoE	84.0±0.5	14.4±1.2	72.3±0.4	66.0±1.1	73.5±0.8	12.1±0.5	64.4±0.6	61.1±2.9
SCE	86.5±0.3	**4.3±0.4**	**82.9±0.4**	**83.9±0.6**	71.7±1.9	**2.5±1.8**	**70.2±0.2**	**73.9±1.0**

4 EXPERIMENTS

4.1 SIMULATION EXPERIMENTS

We evaluate SCE on LIBERO-Goal and LIBERO-Long (Liu et al., 2023) in the main paper, while results on LIBERO-Object and LIBERO-Spatial are provided in Appendix E.3.

Continual learning protocol. Each suite forms a 10-stage stream in benchmark order, following Stellar VLA (Wu et al., 2025). Unless specified otherwise, each stage trains on the new task and five replay demonstrations per old task. We evaluate all observed tasks after each stage, using 50 rollouts per task. Appendices D.1 and D.5 detail the demonstration counts, training hyperparameters, and compute resources.

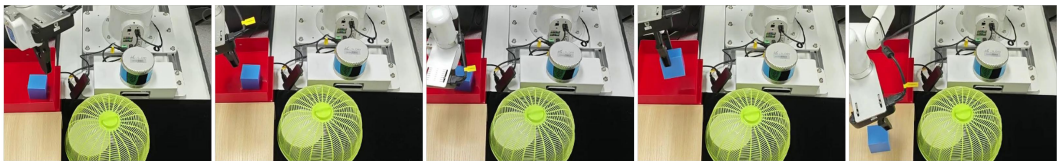
Baselines. We compare with UniVLA, MoDE, T-Stellar, TS-Stellar, IsCiL, and LOTUS using results reported by Stellar VLA (Wu et al., 2025). Our Sequential, ER, and Task-MoE implementations share the same pretrained π_0 backbone as SCE. Sequential and ER isolate replay; Task-MoE assigns one private expert per task. Appendix D.3 details the baseline implementations and result sources.

Metrics. We report Forward Transfer (FWT) on new tasks, Negative Backward Transfer (NBT; lower is better) on old tasks, Area Under the Success Rate Curve (AUC), and average Final Success Rate (Final SR). Appendix D.4 defines these metrics.

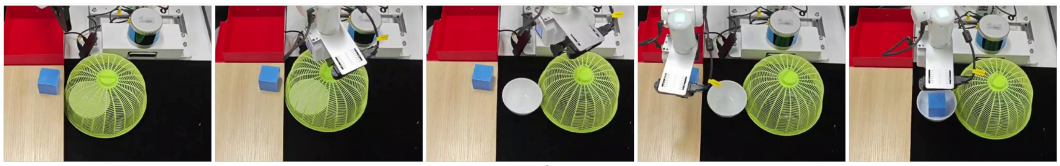
Results. Table 1 shows that SCE achieves the best AUC, Final SR, and NBT on both primary suites. On LIBERO-Goal, it reaches 82.9 AUC and 83.9 Final SR, exceeding Task-MoE by 10.6 and 17.9 points. On LIBERO-Long, it improves AUC from 64.4 to 70.2 and Final SR from 61.1 to 73.9, while reducing NBT from 12.1 to 2.5. The matched-backbone comparison with Task-MoE highlights the value of organizing adaptation around reusable skills under the same replay protocol. Improvements in both AUC and Final SR show that the advantage extends across the learning stream and persists at the final stage, supporting skill composition for retaining previously acquired behaviors as new tasks are introduced. Appendix E.2 provides additional comparisons with published lifelong skill-learning methods.

Replay efficiency. To evaluate replay efficiency, we compare SCE with ER using a larger replay buffer on LIBERO-Goal. Under the same update budget, SCE uses five replay demonstrations per old task and achieves 82.5 AUC and 83.4 Final SR, compared with 77.5 AUC and 79.0 Final SR for ER using 32 demonstrations per old task. Appendix E.4 details the matched-update and matched-epoch comparisons.

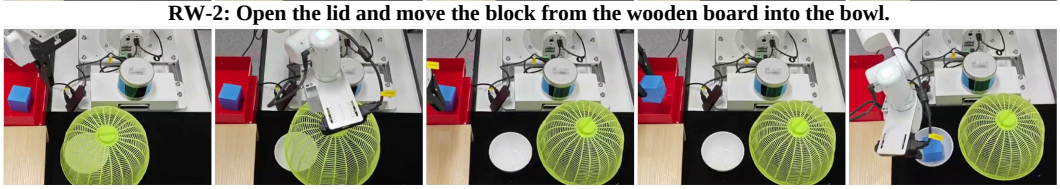
Skill arrival order. We reorder the same ten LIBERO-Goal tasks, keeping the final five-skill set unchanged while varying when new skills enter the skill base. With new skills introduced at stages 6 and 9, SCE achieves 80.7 AUC and 81.4 Final SR, supporting continued learning as the skill base expands. Appendix E.4 reports all three skill-arrival orders and their per-stage skill counts.



RW-1: Move the block from the box onto the wooden board.



RW-2: Open the lid and move the block from the wooden board into the bowl.



RW-3: Open the lid and move the block from the box into the bowl.

Figure 3: Representative snapshots of RW-1, RW-2, and RW-3, ordered from task start to completion.

4.2 Real-world experiments

We evaluate the Mercury B1's 7-DoF arm on three sequential tasks: RW-1 transfers a block from a box to a board; RW-2 opens a lid and transfers the block from the board into a bowl; RW-3 opens the lid and transfers the block from the box into the bowl. These tasks test reuse of retrieval, transfer, placement, and lid-opening skills. Fig. 3 shows representative rollouts; supplementary videos show execution.

Table 2: Real-world continual learning results.

Method	FWT ↑	NBT ↓	AUC ↑	Final SR ↑
Sequential	55.6	45.6	28.7	10.0
ER	52.2	23.9	37.2	22.2
Task-MoE	64.4	8.3	59.8	58.9
SCE	**80.0**	**0.6**	**79.1**	**80.0**

The three tasks share manipulation behaviors while varying the block's source and destination. In particular, RW-2 and RW-3 both require opening the lid and placing the block into the bowl, but retrieve it from different locations. This task stream evaluates reuse of learned behaviors across related manipulation tasks. Appendix B.3 provides the corresponding skill annotations.

Protocol and baselines. SCE, Sequential, ER, and Task-MoE share the pretrained π_0 backbone and three-stage protocol. Replay retains five demonstrations per old task; each observed task is evaluated over 30 trials per method after each stage. Appendix D.2 describes the robot setup, demonstration statistics, and task-specific success criteria.

Results. As shown in Table 2, SCE leads all four metrics, improving Final SR from 58.9 to 80.0 and reducing NBT from 8.3 to 0.6 over Task-MoE. These results show that SCE's advantages also hold on real robots, enabling it to learn new tasks while retaining previously acquired behaviors.

4.3 Analysis of feature drift impact under closed-loop control

SCE preserves old-task representations and rollout behavior. We compare acquisition and final checkpoints on identical demonstration inputs, measuring cosine distance in visual-language representations and decoder features before action projection. Immediate-action drift uses the first action position; horizon-action drift uses hidden features averaged across action positions. Fig. 4(a) shows that Sequential's drift is respectively 1.4, 5.7, and 6.9 times SCE's at these three levels. Fig. 4(b) further shows that SCE maintains smaller rollout state distances throughout execution when both checkpoints start from identical simulator states. Appendix E.6 details the matched-input feature measurements, shared-state rollout protocol, and bootstrap confidence intervals.

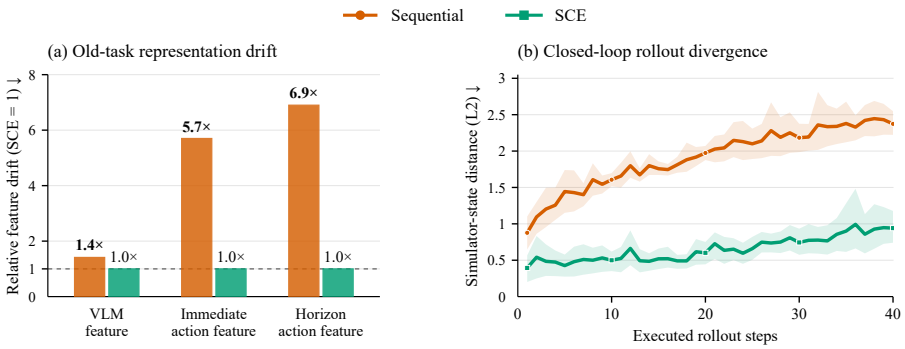


Figure 4: Old-task feature retention and rollout stability. (a) Mean feature cosine distance, normalized to SCE within each feature type. (b) Simulator-state L2 distance between each method's acquisition and final-checkpoint rollouts from shared initial states. Curves show task-averaged distances; shading denotes pointwise 95% bootstrap confidence intervals across tasks. Lower is better in both panels.

4.4 Ablation Experiments

Table 3 evaluates the contributions of CSG and DETE. Compared with SCE w/o CSG, full SCE improves AUC from 66.1 to 82.5 and Final SR from 63.6 to 83.4, demonstrating the benefit of explicitly organizing reusable skills. Appendix E.5 further analyzes CSG segmentation accuracy, naming consistency, and downstream performance with automatic versus verified annotations.

Table 3: Ablations on LIBERO-Goal. EEB/TEB: Execution/Transition Expert Branch.

Method	FWT ↑	NBT ↓	AUC ↑	Final SR ↑
SCE w/o CSG	72.4	7.0	66.1	63.6
SCE w/o TEB	61.4	**-6.3**	67.1	69.4
SCE w/o EEB	80.6	6.5	75.0	80.0
SCE	**86.2**	4.3	**82.5**	**83.4**

Within DETE, adding the Transition Expert Branch improves FWT from 61.4 to 86.2 and AUC from 67.1 to 82.5, while adding the Execution Expert Branch improves AUC from 75.0 to 82.5 and reduces NBT from 6.5 to 4.3. These results support the complementary roles of skill-specific execution and cross-skill transitions in continual task learning.

Across LIBERO-Goal tasks, the Execution-branch fusion weight decreases by 16.2% on average near skill boundaries, while the normalized contribution of the Transition Expert Branch increases in representative transition windows. The analysis in Appendix E.7 shows how adaptive fusion shifts emphasis from skill-specific execution toward transition modeling during skill changes. Appendix E.8 further illustrates full-trajectory expert reuse and the functional stability of an old Execution Expert.

5 Conclusion

We propose SCE, a skill-compositional framework for ECL that constructs a reusable skill base from sequential task demonstrations through CSG and composes these skills for new task learning through DETE. By combining skill-specific execution with adaptive cross-skill transitions, SCE supports stable reuse of learned behaviors across evolving tasks. Experiments on LIBERO benchmarks and real-world manipulation tasks show improved retention and overall continual learning performance. Feature-drift analyses and ablation studies connect these gains to reduced representational drift and coherent skill composition under closed-loop control. A promising future direction is to extend skill-compositional ECL to larger and more open-ended skill repertoires.

AI Use Statement

Generative AI tools were used to assist with language editing and revision of the manuscript.

REPRODUCIBILITY STATEMENT

The main text specifies the continual-learning protocol, metrics, and model objective. The appendix documents CSG, skill annotations, dynamic expansion, baseline implementations, data counts, hyperparameters, and additional results.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. Learning to act anywhere with task-centric latent actions. In *Proceedings of Robotics: Science and Systems*, 2025.

Massimo Caccia, Pau Rodriguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Page-Caccia, Issam Hadj Laradji, Irina Rish, Alexandre Lacoste, David Vázquez, and Laurent Charlin. Online fast adaptation and knowledge accumulation (OSAKA): A new approach to continual learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 16532–16545, 2020.

Francisco M. Castro, Manuel J. Marin-Jimenez, Nicolas Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *European Conference on Computer Vision*, pp. 233–248, 2018.

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

Chongkai Gao, Haichuan Gao, Shangqi Guo, Tianren Zhang, and Feng Chen. CRIL: Continual robot imitation learning via generative and prediction model. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 6747–6754, 2021.

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

Qinghua Hu, Yucong Gao, and Bing Cao. Curiosity-driven class-incremental learning via adaptive sample selection. *IEEE Transactions on Circuits and Systems for Video Technology*, 32(12): 8660–8673, 2022b.

Xinting Hu, Kaihua Tang, Chunyan Miao, Xian-Sheng Hua, and Hanwang Zhang. Distilling causal effect of data in class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3957–3966, 2021.

Ziqi Jia, Anmin Wang, Xiaoyang Qu, Xiaowen Yang, and Jianzong Wang. Hierarchical-task-aware multi-modal mixture of incremental LoRA experts for embodied continual learning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 28415–28427, 2025.

Dahuin Jung, Dongyoon Han, Jihwan Bang, and Hwanjun Song. Generating instance-level prompts for rehearsal-free continual learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11847–11857, 2023.

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

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

Daehee Lee, Dongsu Lee, TaeYoon Kwack, Wonje Choi, and Honguk Woo. Policy compatible skill incremental learning via lazy learning interface. In *Advances in Neural Information Processing Systems*, 2025.

Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information Fusion*, 58:52–68, 2020.

Jiashuo Li, Shaokun Wang, Bo Qian, Yuhang He, Xing Wei, Qiang Wang, and Yihong Gong. Dynamic integration of task-specific adapters for class incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 30545–30555, 2025.

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

Huiwei Lin, Shanshan Feng, Xutao Li, Wentao Li, and Yunming Ye. Anchor assisted experience replay for online class-incremental learning. *IEEE Transactions on Circuits and Systems for Video Technology*, 33(5):2217–2232, 2023.

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

Yaoyao Liu, Bernt Schiele, and Qianru Sun. Adaptive aggregation networks for class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2544–2553, 2021.

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 *Proceedings of the International Conference on Learning Representations*, 2024.

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

German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

Lin Peng, Cong Wan, Shaokun Wang, Xiang Song, Yuhang He, and Yihong Gong. CIA: Class- and instance-aware adaptation for vision-language models. In *Proceedings of the ACM International Conference on Multimedia*, pp. 2870–2879, 2025.

Moritz Reuss, Jyothish Pari, Pulkit Agrawal, and Rudolf Lioutikov. Efficient diffusion transformer policies with mixture of expert denoisers for multitask learning. In *Proceedings of the International Conference on Learning Representations*, 2025.

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

Ralf Römer, Yi Zhang, Yuming Li, and Angela P. Schoellig. CLARE: Continual learning for Vision-Language-Action models via autonomous adapter routing and expansion. *arXiv preprint arXiv:2601.09512*, 2026.

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pp. 627–635, 2011.

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

James Seale Smith, Leonid Karlinsky, Vyshnavi Gutta, Paola Cascante-Bonilla, Donghyun Kim, Assaf Arbelle, Rameswar Panda, Rogerio Feris, and Zsolt Kira. CODA-Prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11909–11919, 2023.

Mingchen Song, Xiang Deng, Guoqiang Zhong, Qi Lv, Jia Wan, Yinchuan Li, Jianye Hao, and Weili Guan. Few-shot vision-language action-incremental policy learning. *arXiv preprint arXiv:2504.15517*, 2025.

Jonathan Spencer, Sanjiban Choudhury, Arun Venkatraman, Brian Ziebart, and J. Andrew Bagnell. Feedback in imitation learning: The three regimes of covariate shift. *arXiv preprint arXiv:2102.02872*, 2021.

Yu-Ming Tang, Yi-Xing Peng, and Wei-Shi Zheng. When prompt-based incremental learning does not meet strong pretraining. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1706–1716, 2023.

Xiaoyu Tao, Xinyuan Chang, Xiaopeng Hong, Xing Wei, and Yihong Gong. Topology-preserving class-incremental learning. In *European Conference on Computer Vision*, pp. 254–270, 2020a.

Xiaoyu Tao, Xiaopeng Hong, Xinyuan Chang, and Yihong Gong. Bi-objective continual learning: Learning "new" while consolidating "known". In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5989–5996, 2020b.

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

Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. DualPrompt: Complementary prompting for rehearsal-free continual learning. In *European Conference on Computer Vision*, pp. 631–648, 2022a.

Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 139–149, 2022b.

Guile Wu, Shaogang Gong, and Pan Li. Striking a balance between stability and plasticity for class-incremental learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1124–1133, 2021.

Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 374–382, 2019.

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

Jingkai Xu and Xiangli Nie. SPECI: Skill prompts based hierarchical continual imitation learning for robot manipulation. *IEEE Transactions on Cognitive and Developmental Systems*, 18(2):488–503, 2026.

Shipeng Yan, Jiangwei Xie, and Xuming He. DER: Dynamically expandable representation for class incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3014–3023, 2021.

Yuanqi Yao, Siao Liu, Haoming Song, Delin Qu, Qizhi Chen, Yan Ding, Bin Zhao, Zhigang Wang, Xuelong Li, and Dong Wang. Think small, act big: Primitive prompt learning for lifelong robot manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22573–22583, 2025.

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.

Wenbo Zhang, Tianrun Hu, Hanbo Zhang, Yanyuan Qiao, Yuchu Qin, Yang Li, Jiajun Liu, Tao Kong, Lingqiao Liu, and Xiao Ma. Chain-of-Action: Trajectory autoregressive modeling for robotic manipulation. In *Advances in Neural Information Processing Systems*, 2025b.

Hanbin Zhao, Yongjian Fu, Mintong Kang, Qi Tian, Fei Wu, and Xi Li. MgSvF: Multi-grained slow versus fast framework for few-shot class-incremental learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(3):1576–1588, 2024.

Zexin Zheng, Jia-Feng Cai, Xiao-Ming Wu, Yi-Lin Wei, Yu-Ming Tang, Ancong Wu, and Wei-Shi Zheng. iManip: Skill-incremental learning for robotic manipulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 13890–13900, 2025.

Da-Wei Zhou, Hai-Long Sun, Han-Jia Ye, and De-Chuan Zhan. Expandable subspace ensemble for pre-trained model-based class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 23554–23564, 2024.

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

APPENDIX CONTENTS

A. CSG algorithm and prompt 15
A.1. Phase 1: State-Aware Task Decomposition 15
A.2. Phase 2: VLM-Grounded Skill Induction 16
B. Skill base and annotation examples 16
B.1. Simulation skill base 16
B.2. Simulation annotation 17
B.3. Real-world skill base 18
C. DETE implementation details 18
C.1. Dynamic skill-space expansion 19
D. Experimental details 19
D.1. Simulation setup 19
D.2. Real-world robot setup 20
D.3. Baselines 20
D.4. Evaluation metrics 21
D.5. Training hyperparameters and compute 21
E. Additional experiments and analyses 21
E.1. Skill-supervision hyperparameter 21
E.2. Direct comparisons with lifelong skill-learning methods 22
E.3. LIBERO-Object and LIBERO-Spatial 22
E.4. Large replay and delayed skill arrival 23
E.5. CSG annotation reliability 24
E.6. Feature-drift protocol 25
E.7. Adaptive balancing at skill transitions 25
E.8. Full-trajectory routing and old-expert stability 25
F. Limitations and future work 26
G. Broader impacts 26

A CSG ALGORITHM AND PROMPT

This section provides the implementation details of Compositional Skill Grounding (CSG) used in Sec. 3.3. CSG converts each demonstration into temporally coherent segments and grounds each segment into a reusable skill. CSG is used for LIBERO simulation data (Liu et al., 2023) and real-world demonstrations. Consistent with the main text, CSG proceeds in two phases: **Phase 1** partitions each demonstration into temporally coherent segments according to robot-state dynamics, and **Phase 2** grounds these segments with a VLM to induce reusable skills and update a skill base $\mathcal{B}$.

Algorithm 1 Compositional Skill Grounding

Require: Demonstration stream $\mathcal{D}$, VLM $\mathcal{M}$, segmentation parameters Ω
Ensure: Skill base $\mathcal{B}$ and frame-level skill annotations $\mathcal{S}$

$\mathcal{B} \leftarrow \emptyset, \mathcal{S} \leftarrow \emptyset$
for each demonstration $\mathcal{X} = (\mathcal{O}, \mathcal{R}, l) \in \mathcal{D}$ **do**
  Phase 1: State-aware task decomposition
    $B_g \leftarrow \textsc{DetectGripperChanges}(\mathcal{R}, \Omega)$ ▷ Rule 1: gripper-state change
    $B_e \leftarrow \textsc{DetectTrendChanges}(\mathcal{R}, \Omega)$ ▷ Rule 2: end-effector trend change
    $B \leftarrow \textsc{MergeAndFilterBoundaries}(B_g, B_e, \Omega)$
    $\{\mathcal{X}^{(m)}\}_{m=1}^{M} \leftarrow \textsc{SplitDemonstration}(\mathcal{X}, B)$
  Phase 2: VLM-grounded skill induction
    for $m = 1, \ldots, M$ **do**
      $\hat{s}^{(m)} \leftarrow \mathcal{M}(\mathcal{O}^{(m)}, l, \mathcal{B})$ ▷ Ground segment with current skill base
      if $\textsc{Match}(\hat{s}^{(m)}, \mathcal{B}) = \emptyset$ **then**
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{\hat{s}^{(m)}\}$
        $s^{(m)} \leftarrow \hat{s}^{(m)}$
      else
        $s^{(m)} \leftarrow \textsc{Match}(\hat{s}^{(m)}, \mathcal{B})$
      end if
    end for
    Assign $s_i = s^{(m)}$ for every frame i inside segment $\mathcal{X}^{(m)}$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{s_i\}_{i=1}^{F}$
end for
return $\mathcal{B}, \mathcal{S}$

A.1 PHASE 1: STATE-AWARE TASK DECOMPOSITION

Let $\mathcal{X} = (\mathcal{O}, \mathcal{R}, l)$ denote a raw task demonstration of temporal length F, where $\mathcal{O} = \{o_i\}_{i=1}^{F}$ is the observation sequence, $\mathcal{R} = \{r_i\}_{i=1}^{F}$ is the robot-state sequence, and l is the language instruction. Inspired by the keyframe intuition of Chain-of-Action (Zhang et al., 2025b), CSG defines two state-based rules for temporal decomposition:

- **Rule 1: Gripper-state change.** CSG marks a temporal boundary when the gripper state changes between adjacent timesteps. In implementation, this is detected by smoothing the gripper openness and identifying open/closed state switches.
- **Rule 2: End-effector trend change.** CSG marks a temporal boundary when the end-effector motion trend changes abruptly. In implementation, this is measured by the cosine similarity between smoothed pre-boundary and post-boundary motion vectors.

CSG applies Rule 1 and Rule 2 to the robot-state sequence $\mathcal{R}$ and obtains a set of temporal boundary indices. Nearby boundaries are merged, boundaries inside stable closed-contact periods are filtered, and very short segments are merged into adjacent segments. These indices split the demonstration into M contiguous temporal segments $\{\mathcal{X}^{(m)}\}_{m=1}^{M}$. Each segment is defined as $\mathcal{X}^{(m)} = (\mathcal{O}^{(m)}, \mathcal{R}^{(m)}, l)$, where $\mathcal{O}^{(m)}$ and $\mathcal{R}^{(m)}$ denote the corresponding observation and robot-state subsequences. Algorithm 1 summarizes the complete CSG procedure used to produce the skill base and frame-level annotations.

Table 4 summarizes the segmentation settings. The rule structure is shared across domains, while the numerical values differ because the simulation and real-world streams use different sampling rates and gripper-state conventions.

A.2 Phase 2: VLM-Grounded Skill Induction

After temporal decomposition, CSG processes the temporal segments $\{\mathcal{X}^{(m)}\}_{m=1}^{M}$ sequentially using the current skill base $\mathcal{B}$. For each segment $\mathcal{X}^{(m)}$, Qwen3-VL-Plus (Bai et al., 2025) grounds the segment observation $\mathcal{O}^{(m)}$ and language instruction l with reference to $\mathcal{B}$, and proposes a candidate skill $\hat{s}^{(m)}$. The candidate is then matched against $\mathcal{B}$. If no existing skill matches $\hat{s}^{(m)}$, the candidate is inserted as a new reusable skill; otherwise, the matched skill is reused. The selected skill is denoted as $s^{(m)}$. A candidate skill is matched to an existing skill only when both the visible manipulation intent and the source/target relation are consistent with the existing skill definition. After all segments are processed, CSG obtains the updated skill base $\mathcal{B}$ and the segment-level assignments $\{(\mathcal{X}^{(m)}, s^{(m)})\}_{m=1}^{M}$.

The segment-level assignments are converted into the frame-level annotations used for training. Let $\mathcal{S} = \{s_i\}_{i=1}^{F}$ denote the annotation sequence for demonstration $\mathcal{X}$. For any frame i inside segment $\mathcal{X}^{(m)}$, we set $s_i = s^{(m)}$. Simulation segment grounding samples the full episode video at 8 FPS, while real-world segment grounding samples the aligned video at 6 FPS. Very short segments are temporally stretched before upload so that the action intent is visible.

Prompt: VLM grounding for CSG

Role. Robot Action Semantics Expert.
Task. Ground each predefined robot video segment into a reusable semantic skill.
Inputs.

- Current skill base `[SKILL_BASE_JSON]`.
- Task language instruction `[LANGUAGE]`.
- Predefined segment observations `[SEGMENT_VIDEOS]`, where each item contains the sampled video frames for one segment.

Instruction. The video has already been segmented by robot-state dynamics. Do not change the segment boundaries. For each listed segment observation, use the visible segment content and the task language instruction to decide whether it matches an existing skill in the skill base. If it does not match any existing skill, create a concise semantic skill name and add it as a new skill. Otherwise, reuse the matched skill name.
Rules.

1. Ground each segment by visible manipulation intent.
2. Reuse an existing skill only when the action intent and source/target relation match the current segment.
3. Create a new skill only when no existing skill describes the segment.
4. Do not use pure direction or trajectory names such as `LeftMove`, `RightMove`, `DownLeft`, or `Orbit`; use motion direction only as secondary evidence.
5. Output every listed segment exactly once and return valid JSON only.

Output. A JSON object with a `segments` list and a `skill_base_updates` list. Each segment item contains `segment`, `skill_name`, `is_new_skill`, and `semantic_reasoning`; each skill-base update contains `skill_name` and `definition`.

B Skill base and annotation examples

This section reports the named skill vocabulary and the resulting task-to-skill decompositions. The names are the natural semantic labels produced by the VLM grounding step.

B.1 Simulation skill base

Table 5 lists the induced semantic skill vocabulary for the two simulation streams.

Table 4: CSG segmentation settings for simulation and real-world data.

Setting	LIBERO simulation	Real world
Input stream	Synchronized simulation demonstrations	Synchronized real-world demonstrations
Main video	Third-view RGB frames	Aligned third-view RGB video
Output frame rate	20 FPS	10 FPS
Gripper openness	Sum of absolute final two gripper states	Final normalized gripper state
Threshold quantiles	0.10, 0.90	0.10, 0.90
Default gripper threshold	0.04	0.50
Hysteresis	Not used	Default margin 0.08
Majority window	7 frames	5 frames
Minimum contact run	5 frames	4 frames
Trend smoothing window	7 frames	5 frames
Trend lookaround	6 frames	4 frames
Trend cosine threshold	0.35	0.35
Minimum step norm	0.0015	0.0025
Minimum local displacement	Not used	0.010
Minimum boundary gap	12 frames	10 frames
Drop trend changes inside closed contact	Yes	Yes
Closed-contact check window	8 frames	6 frames
Minimum segment length	8 frames	8 frames

Table 5: Natural semantic skill vocabularies induced by CSG.

Dataset stream	Semantic skill name	Shared semantic summary
LIBERO-Goal	TargetManipulate	Move the current target object or articulated part toward its goal configuration.
	SettleObject	Push, release, or settle the manipulated object/component after contact.
	AcquireObject	Approach and secure an object or handle before the next manipulation step.
	EngageObject	Bring the gripper into contact with an object, surface, or control to start interaction.
	FineManipulate	Execute constrained adjustments such as knob operation or precise object placement.
LIBERO-Long	TransferObject	Transport a grasped object between workspace regions or toward the next receptacle.
	InsertObject	Place an object into a container, compartment, drawer, or microwave interior.
	PickObject	Approach and grasp a tabletop object for a subsequent transfer.
	PlaceInContainer	Move a held object into a basket, plate, stove area, or other target support.
	ReachObject	Reach toward a target object or handle to initiate manipulation.
	FineManipulate	Perform constrained operations such as closing, turning, or precision adjustment.

B.2 Simulation annotation

Tables 6 and 7 report the task-to-skill annotations for LIBERO-Goal and LIBERO-Long, respectively.

Table 6: Natural semantic CSG annotations for LIBERO-Goal. Task index (0–9) specifies the benchmark task order.

Task index	Task	Semantic skill sequence
0	open the middle drawer of the cabinet	TargetManipulate → SettleObject
1	put the bowl on the stove	AcquireObject → EngageObject
2	put the wine bottle on top of the cabinet	EngageObject → TargetManipulate
3	open the top drawer and put the bowl inside	TargetManipulate → SettleObject → AcquireObject → TargetManipulate
4	put the bowl on top of the cabinet	AcquireObject → TargetManipulate
5	push the plate to the front of the stove	AcquireObject → SettleObject
6	put the cream cheese in the bowl	EngageObject → FineManipulate
7	turn on the stove	EngageObject → FineManipulate
8	put the bowl on the plate	AcquireObject → SettleObject
9	put the wine bottle on the rack	EngageObject → TargetManipulate

Table 7: Natural semantic CSG annotations for LIBERO-Long. Task index (0–9) specifies the benchmark task order.

Task index	Task	Semantic skill sequence
0	put both the alphabet soup and the tomato sauce in the basket	PickObject → PlaceInContainer → InsertObject → PlaceInContainer
1	put both the cream cheese box and the butter in the basket	PickObject → PlaceInContainer → InsertObject → TransferObject
2	turn on the stove and put the moka pot on it	TransferObject → FineManipulate → ReachObject → TransferObject
3	put the black bowl in the bottom drawer of the cabinet and close it	ReachObject → TransferObject → FineManipulate
4	put the white mug on the left plate and put the yellow and white mug on the right plate	ReachObject → PickObject → PlaceInContainer → TransferObject
5	pick up the book and place it in the back compartment of the caddy	TransferObject → InsertObject
6	put the white mug on the plate and put the chocolate pudding to the right of the plate	PickObject → ReachObject → FineManipulate
7	put both the alphabet soup and the cream cheese box in the basket	ReachObject → PlaceInContainer → ReachObject → TransferObject
8	put both moka pots on the stove	ReachObject → PickObject → TransferObject → PickObject
9	put the yellow and white mug in the microwave and close it	ReachObject → TransferObject → InsertObject → TransferObject

B.3 REAL-WORLD SKILL BASE

Tables 8 and 9 report the real-world skill vocabulary and task-level annotations.

Table 8: Natural semantic real-world skill vocabulary induced from CSG segments.

Skill name	Segment source	Semantic summary
AcquireObject	Initial approach-and-secure action	Move to the target item and secure it for subsequent motion.
TransferObject	Carrying action after acquisition	Move a held item from its source region toward a target support.
OpenManipulate	Opening action	Move a removable cover away from its closed configuration.
LowerManipulate	Lowering action	Return a removable cover toward the workspace.
DepositObject	Final release action	Move the held item to the target receptacle or support and release it.

Table 9: Natural semantic CSG annotations for real-world tasks.

Task label	Task	Semantic skill sequence
RW-1	put_the_block_from_the_box_on_the_wooden_board	AcquireObject → TransferObject
RW-2	open_the_lid_and_put_the_block_from_the_wooden_board_in_the_bowl	OpenManipulate → LowerManipulate → AcquireObject → DepositObject
RW-3	open_the_lid_and_put_the_block_from_the_box_in_the_bowl	OpenManipulate → LowerManipulate → AcquireObject → DepositObject

C DETE IMPLEMENTATION DETAILS

SCE is built on a pretrained π_0 VLA model (Black et al., 2024). During continual learning, DETE is trainable and the vision-language backbone is adapted with LoRA (Hu et al., 2022a), while the main action decoder is frozen. DETE provides a decoder-side adaptation path for skill composition without updating the original action-decoder weights. During both training and inference, DETE routes experts using the predicted skill distribution $\mathbf{p}$ obtained from the observation, instruction, and robot state; the CSG annotations are used only to supervise the skill predictor during training.

Each DETE expert is implemented as a low-rank residual adapter inserted into the action decoder attention projections and feed-forward block. Specifically, the adapters are inserted into the $q/k/v/o$

attention projections and the FFN. The Transition Expert Branch uses $N = 4$ experts in all experiments, with rank 32 for each Transition Expert. The Execution Expert Branch uses one expert for each skill in the CSG skill base, with rank 8 for each Execution Expert: $K = 5$ for LIBERO-Goal, $K = 6$ for LIBERO-Long, $K = 2$ for LIBERO-Object, $K = 5$ for LIBERO-Spatial, and $K = 5$ for the real-world task stream.

The training objective follows the main paper:

$$\mathcal{L} = \mathcal{L}_{\text{flow}} + \lambda \mathcal{L}_{\text{skill}},$$

where $\mathcal{L}_{\text{flow}}$ is the original action-generation loss and $\mathcal{L}_{\text{skill}}$ is the skill supervision loss for the CSG-derived skill label. We use $\lambda = 0.2$ for both simulation and real-world experiments. The analysis of the hyperparameter λ is provided in Sec. E.1. Table 10 summarizes the DETE implementation settings used in all experiments.

Table 10: DETE implementation details.

Item	Setting
Base model	Pretrained π_0 VLA
Trainable modules	DETE and vision-language backbone LoRA
Frozen modules	Main action decoder
Expert insertion	Attention $q/k/v/o$ projections and FFN
Transition Experts	$N = 4$, rank 32
Execution Experts	One per discovered skill; rank 8
Skill loss weight	$\lambda = 0.2$

C.1 Dynamic skill-space expansion

SCE does not require the final number of skills in advance. At stage t, it maintains an append-only skill base $\mathcal{B}_t$ of size $K_t = |\mathcal{B}_t|$. If CSG identifies m new skills at stage $t + 1$, their indices are appended and all existing indices remain unchanged. For an output layer with hidden width d, the classifier is expanded as

$$W_{t+1} = \begin{bmatrix} W_t \\ W_{\text{new}} \end{bmatrix}, \qquad b_{t+1} = \begin{bmatrix} b_t \\ b_{\text{new}} \end{bmatrix}, \tag{8}$$

where $W_t \in \mathbb{R}^{K_t \times d}$ and only the new rows are initialized. The Execution Expert pool is expanded by appending $E_{K_t+1}, \ldots, E_{K_t+m}$. Since the transition router consumes the skill distribution, its projection is expanded by appending the corresponding input columns:

$$W_{\text{p},t+1} = [W_{\text{p},t} \quad W_{\text{p,new}}]. \tag{9}$$

The four Transition Experts are shared and remain fixed in number.

Previously learned Execution Experts remain trainable. Hard top-1 routing sends each sample through one Execution Expert, so an unselected expert receives no action-loss gradient from that sample. Current-task samples update a previous expert when they reuse its skill; replay samples maintain experts for skills that occur only in earlier tasks.

D Experimental details

D.1 Simulation setup

We evaluate on four LIBERO task suites (Liu et al., 2023): LIBERO-Goal, LIBERO-Long, LIBERO-Object, and LIBERO-Spatial.

LIBERO-Goal is a 10-task goal-conditioned suite designed to test procedural knowledge transfer. Its tasks share the same objects and fixed spatial relationships, while the goal specification changes across tasks; the policy must therefore learn different manipulation behaviors under a controlled scene layout. Following the continual evaluation setting used by Stellar VLA (Wu et al., 2025), we organize LIBERO-Goal as a 10-stage stream with one new task per stage. After removing no-op

steps and converting the data used by OpenPI, the ten tasks contain 43, 48, 47, 36, 46, 33, 40, 50, 49, and 36 training episodes, respectively (428 total). At stage t, the model is trained on the current task demonstrations and five replay demonstrations retained from each previously learned task. After each stage, the checkpoint is evaluated on all observed LIBERO-Goal tasks with 50 rollout episodes per task.

LIBERO-Long is the 10-task long-horizon subset split from LIBERO-100 for downstream lifelong-learning evaluation. Its tasks require multi-step object interactions, such as sequential pick-and-place, insertion, closing, and appliance manipulation. We use the same 10-stage continual learning protocol for LIBERO-Long: one long-horizon task is introduced at each stage. The converted tasks contain 33, 49, 41, 35, 38, 41, 36, 43, 29, and 34 training episodes, respectively (379 total). Training uses the current task demonstrations together with five retained replay demonstrations from each previous task. After each stage, the checkpoint is evaluated on all observed LIBERO-Long tasks with 50 rollout episodes per task.

Success rates from these rollouts form the stage-task matrix used by all continual learning metrics.

D.2 Real-world robot setup

The real-world experiments use a single 7-DoF arm on the Mercury B1 robot. The workspace contains a box, a wooden board, a bowl with a removable lid, and a movable block. A fixed third-view Intel RealSense D435 RGB-D camera records each demonstration and rollout. For each real-world task, we collect 50 human teleoperation demonstrations using a MyController-S570 controller. The three tasks form a three-stage stream:

- **RW-1, Box-to-Board.** Move the block from the box to the wooden board. A trial is successful if the block is stably released on the wooden board without dropping outside the workspace.
- **RW-2, Board-to-Bowl.** Open the lid and move the block from the wooden board into the bowl. A trial is successful if the lid is opened sufficiently, the block is released inside the bowl, and the bowl remains upright.
- **RW-3, Box-to-Bowl.** Open the lid and move the block from the box into the bowl. The same success criterion as RW-2 is used, with the source region changed to the box.

Table 11 summarizes the demonstration count, frame count, duration, and episode length for each real-world task.

Table 11: Real-world dataset statistics.

Task ID	Task	Demos	Frames	Duration	Episode length
RW-1	Box-to-Board	50	7,478	12.5 min	149.6 frames avg.
RW-2	Board-to-Bowl	50	12,627	21.0 min	252.5 frames avg.
RW-3	Box-to-Bowl	50	14,205	23.7 min	284.1 frames avg.
Total	–	150	34,310	57.2 min	228.7 frames avg.

At each real-world stage, the model is trained on the 50 demonstrations of the current task and five replay demonstrations retained from each previously learned task. Each real-world task is evaluated over 30 trials per method after each continual-learning stage. A trial is terminated and counted as failed if it exceeds 120 seconds or produces unsafe behavior. The reported success rate is the percentage of successful trials.

D.3 Baselines

All implemented baselines use the same pretrained π_0 backbone (Black et al., 2024) and the same continual learning task order.

- **Sequential.** Fine-tunes on the current task only, without replay. It measures the forgetting caused by plain sequential adaptation.
- **ER** (Rolnick et al., 2019). Uses the same training setup as Sequential, but adds five replay demonstrations for each previous task at every new stage.

- **Task-MoE** (Jia et al., 2025). Uses four shared experts and one private expert for each task under the same backbone and replay protocol. Samples activate the private expert associated with their task.
- **IsCiL** (Lee et al., 2024) and **LOTUS** (Wan et al., 2024). Table 1 uses their LIBERO-90-pretrained results reported by Stellar VLA (Wu et al., 2025).

For UniVLA (Bu et al., 2025), MoDE (Reuss et al., 2025), T-Stellar, and TS-Stellar, Table 1 uses the cross-embodiment-pretrained results reported by Stellar VLA (Wu et al., 2025). All entries in Table 1, including our implementations and the reported baselines, are means and standard deviations over three training seeds.

D.4 Evaluation metrics

Let $R_{t,k}$ denote the success rate after training on $\mathcal{D}^t$ and evaluating on the task set $\mathcal{T}^k$ introduced at stage k, where $1 \le k \le t \le T$. In our experimental streams, each stage introduces one task, so $R_{t,k}$ is the success rate of the task introduced at stage k after the model has learned through stage t. We compute all metrics from this lower-triangular stage-task success matrix R, following the task-trajectory aggregation used in lifelong imitation learning (Zhang et al., 2025a).

$$\begin{aligned}
\mathrm{FWT} &= \frac{1}{T}\sum_{k=1}^{T} R_{k,k}, \\
\mathrm{NBT} &= \frac{1}{T}\sum_{k=1}^{T-1} \frac{1}{T-k} \sum_{t=k+1}^{T} \left(R_{k,k} - R_{t,k}\right), \\
\mathrm{AUC} &= \frac{1}{T}\sum_{k=1}^{T} \frac{1}{T-k+1} \sum_{t=k}^{T} R_{t,k}, \\
\text{Final SR} &= \frac{1}{T}\sum_{k=1}^{T} R_{T,k}.
\end{aligned} \tag{10}$$

FWT measures the average success rate when each task is first learned. NBT measures the average degradation over later stages; lower values indicate less forgetting, and negative values indicate positive backward transfer. AUC first averages each task over its available evaluation trajectory, then averages these task-level trajectories. Final SR is the average success rate over all tasks after the final stage T.

D.5 Training hyperparameters and compute

Table 12 summarizes the default training configuration. We use the same configuration for SCE and the implemented π_0 baselines (Black et al., 2024) unless a method-specific component is explicitly required.

E Additional experiments and analyses

E.1 Skill-supervision hyperparameter

We analyze the effect of λ, which balances the skill supervision loss and the flow-matching loss. The sweep in Fig. 5 evaluates AUC under different values of λ.

The results show a clear trade-off. Without skill supervision ($\lambda = 0$), AUC drops to 66.1, indicating that action fitting alone does not provide enough pressure to organize the DETE experts around reusable CSG skills. Increasing λ to 0.1 improves AUC to 79.7, and $\lambda = 0.2$ achieves the best AUC of 82.5. A slightly larger value ($\lambda = 0.3$) remains competitive at 79.6, but further increasing λ to 0.5 reduces AUC to 60.4. This degradation is consistent with the objective in the main paper: the skill supervision loss should guide expert routing and skill alignment, but it should not dominate the action-generation objective. Therefore, we use $\lambda = 0.2$ as the default value for both simulation and real-world experiments.

Table 12: Default training hyperparameters and compute resources.

Item	Setting
Base policy	Pretrained π_0 VLA
Optimizer	AdamW
Learning rate	1×10^{-4}
Weight decay	0.01
Training epochs	20
Batch size	16
Scheduler	1000-step linear warmup + cosine decay
Action chunk horizon	$H = 10$
Replay memory	5 demonstrations per previous task
Skill loss weight	$\lambda = 0.2$
Evaluation	50 rollout episodes per task in simulation; 30 physical trials per task and method
Compute workers	4 NVIDIA A100-SXM4-40GB GPUs
Peak GPU memory	Approximately 30 GiB per GPU

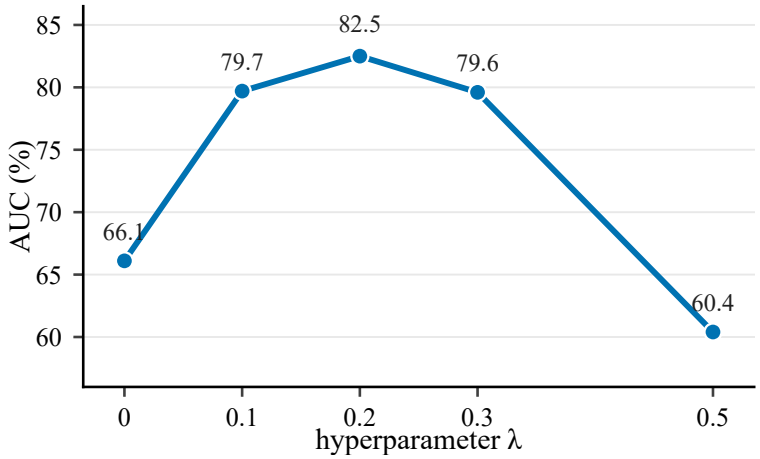


Figure 5: AUC under different values of λ.

E.2 Direct comparisons with lifelong skill-learning methods

Table 13 compares SCE with M2Distill and T2S on LIBERO-Goal. The M2Distill and T2S results are taken from Table I of T2S (Zhang et al., 2025a), using their original policy architectures and training protocols. All entries are means and standard errors over three training seeds. For SCE, we use the same runs as Table 1 and convert the standard deviations to standard errors by dividing by $\sqrt{3}$.

Table 13: Comparison with M2Distill and T2S on LIBERO-Goal. Results are mean $\pm$ standard error over three training seeds.

Method	FWT ↑	NBT ↓	AUC ↑
M2Distill (Roy et al., 2025) (reported)	71.0±1.0	20.0±3.0	57.0±2.0
T2S (Zhang et al., 2025a) (reported)	84.0±1.0	**−1.0±2.0**	70.0±2.0
SCE	**86.5±0.2**	4.3±0.2	**82.9±0.3**

E.3 LIBERO-Object and LIBERO-Spatial

LIBERO-Object changes object identity while retaining a shared high-level program; CSG obtains two reusable skills. LIBERO-Spatial changes initial locations, support relations, and transport

Table 14: Ten-stage continual-learning results on the additional LIBERO suites. Results are mean ± sample standard deviation over three independent training seeds.

Suite	Method	FWT ↑	NBT ↓	AUC ↑	Final SR ↑
Object	Sequential	88.7±0.6	71.1±0.0	33.3±0.7	20.0±1.7
	ER	89.7±1.3	18.9±1.6	74.2±0.1	73.8±1.3
	Task-MoE	92.5±1.0	3.6±1.3	89.5±0.5	91.4±0.5
	SCE	**93.3±0.3**	**3.5±1.1**	**90.6±0.4**	**92.6±0.4**
Spatial	Sequential	90.3±0.7	81.2±0.9	26.3±0.1	9.0±0.2
	ER	89.3±1.7	46.8±1.9	51.3±1.1	33.7±1.4
	Task-MoE	90.6±0.5	7.2±0.4	85.1±0.4	85.9±0.3
	SCE	**92.4±0.3**	**6.1±0.9**	**87.8±0.3**	**87.2±0.2**

geometry; CSG obtains five skills that form five task-level compositions. Across three seeds, SCE obtains the strongest mean result on all four metrics for both suites.

E.4 Large replay and delayed skill arrival

We evaluate replay efficiency on LIBERO-Goal by comparing SCE, which retains five demonstrations per old task, with the same π_0 ER implementation retaining 32 (ER-32). SCE trains for 20 epochs per stage, totaling 102,900 optimizer updates across the ten-stage task stream. We evaluate ER-32 under two training budgets:

Matched updates. ER-32 matches SCE's number of optimizer updates at each stage, giving both methods 102,900 updates in total. This comparison evaluates the two methods under the same update budget with different replay buffer sizes.

Matched epochs. ER-32 trains for 20 epochs per stage, as does SCE. Each epoch passes through the current-task data and the retained replay data. ER-32's larger replay buffer therefore requires more updates per epoch, yielding 294,040 updates over the full task stream, or 2.86 times SCE's total.

Table 15: Large-replay comparison on LIBERO-Goal. Each entry is from one complete continual-learning run.

Method	Replay/task	Updates	FWT	NBT	AUC	Final SR
ER-32 (matched updates)	32	102.9k	74.4	-3.6	77.5	79.0
ER-32 (matched epochs)	32	294.0k	77.0	-12.4	87.6	89.8
SCE	5	102.9k	86.2	4.3	82.5	83.4

At the final stage, ER-32 retains 288 old-task episodes, whereas SCE retains 45. The 20-epoch ER-32 stream uses 2.86 times as many updates and approximately 167.2 A100 GPU-hours on two A100 GPUs. The results characterize two operating regimes: SCE leads in FWT, AUC, and Final SR under matched updates and bounded history, while high-coverage replay provides stronger retention and final performance with substantially more data access and optimization.

To test whether SCE depends on discovering the complete skill base early, we use the same ten LIBERO-Goal tasks and final five-skill set under three orders. Using the task indices in Table 6, the complete task sequences are $0 \to 1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7 \to 8 \to 9$ for Benchmark, $0 \to 3 \to 5 \to 8 \to 4 \to 1 \to 2 \to 9 \to 6 \to 7$ for Delayed, and $0 \to 6 \to 2 \to 3 \to 8 \to 4 \to 7 \to 9 \to 5 \to 1$ for Random (permutation seed 2026).

The delayed order introduces new skills at stages 6 and 9 and retains low forgetting and strong overall performance. The random seed here generates the task permutation; it is separate from the three independent training runs used for the main LIBERO-Goal uncertainty estimates.

Table 16: Skill-base growth and performance under different task orders. Each order is evaluated in one complete continual-learning run.

Order	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9	K_{10}	FWT	NBT	AUC	Final SR
Benchmark	2	4	4	4	4	4	5	5	5	5	86.2	4.3	82.5	83.4
Delayed	2	3	3	3	3	4	4	4	5	5	83.8	5.8	80.7	81.4
Random (seed 2026)	2	4	4	5	5	5	5	5	5	5	88.2	4.6	82.9	86.2

E.5 CSG ANNOTATION RELIABILITY

CSG produces accurate temporal segments and stable semantic names on LIBERO-Goal. We evaluate segmentation against verified reference annotations on all 428 training episodes and assess naming stability through repeated VLM queries for fixed segments.

Boundary localization. Let P_{CSG} and P_{ref} denote the sets of predicted and reference boundary positions. We match boundaries within the same episode using a tolerance of ten frames, equivalent to 0.5 seconds at 20 FPS. Each predicted or reference boundary can participate in at most one match. Let M_{10} be the number of matched boundaries. Boundary precision P_{10}, recall R_{10}, and F1 are

$$\begin{aligned} P_{10} &= \frac{M_{10}}{|P_{\mathrm{CSG}}|}, \qquad R_{10} = \frac{M_{10}}{|P_{\mathrm{ref}}|}, \\ \mathrm{F1@10} &= \frac{2P_{10}R_{10}}{P_{10}+R_{10}} = \frac{2M_{10}}{|P_{\mathrm{CSG}}|+|P_{\mathrm{ref}}|} = 87.9\%. \end{aligned} \tag{11}$$

This measure accounts for both extra predicted boundaries and missed reference boundaries.

Temporal segment overlap. Temporal IoU measures agreement over the complete extent of a segment. For a predicted segment $S_{\mathrm{CSG}} = [s_c, e_c]$ and a reference segment $S_{\mathrm{ref}} = [s_r, e_r]$, both represented as inclusive frame intervals, their intersection length I and union length U are

$$\begin{aligned} I &= \max\{0, \min(e_c, e_r) - \max(s_c, s_r) + 1\}, \\ U &= (e_c - s_c + 1) + (e_r - s_r + 1) - I. \end{aligned} \tag{12}$$

We pair predicted and reference segments in temporal order, using each segment at most once and maximizing the total temporal IoU. For the resulting M matched pairs, the mean matched temporal IoU is

$$\begin{aligned} \mathrm{tIoU}(S_{\mathrm{CSG}}, S_{\mathrm{ref}}) &= \frac{I}{U}, \\ \mathrm{mIoU}_{\mathrm{matched}} &= \frac{1}{M}\sum_{m=1}^{M} \mathrm{tIoU}\big(S_{\mathrm{CSG}}^{(m)}, S_{\mathrm{ref}}^{(m)}\big) = 95.9\%. \end{aligned} \tag{13}$$

Together, Boundary F1@10 and matched temporal IoU assess transition localization and agreement over full segment durations.

Semantic naming consistency. For each fixed segment, we independently query Qwen3-VL-Plus three times, keeping the video, task instruction, candidate skill vocabulary, and generation settings unchanged. Let $n_i^{(r)}$ be the skill name from query r for segment i. We encode only the generated skill name. A fixed preprocessing function g splits CamelCase names into words, for example, `TargetManipulate` becomes `Target Manipulate`. The encoder f is the fixed `sentence-transformers/all-mpnet-base-v2` model, producing 768-dimensional features. We use revision `e8c3b32edf5434bc2275fc9bab85f82640a19130`. The normalized name embedding is

$$\mathbf{e}_i^{(r)} = \frac{f\big(g(n_i^{(r)})\big)}{\big\|f\big(g(n_i^{(r)})\big)\big\|_2}, \qquad r \in \{1,2,3\}. \tag{14}$$

For each segment, we average the cosine similarities of all three pairs of names. Since the embeddings have unit norm, cosine similarity equals their inner product. The score for segment i and the mean over N segments are

$$c_i = \frac{1}{3} \sum_{1 \le r < q \le 3} \langle \mathbf{e}_i^{(r)}, \mathbf{e}_i^{(q)} \rangle, \qquad C_{\text{name}} = \frac{1}{N} \sum_{i=1}^{N} c_i = 98.4\%. \tag{15}$$

This score measures the semantic stability of repeated names for the same segment without requiring exact string agreement or a similarity threshold.

Downstream effect of annotation quality. SCE trained with CSG labels achieves AUC and Final SR within 1.5 and 1.8 percentage points of training with verified reference labels. We train SCE once with each label source, using the same training setting and random seed. Table 17 reports all four metrics.

Table 17: Effect of skill annotation quality on LIBERO-Goal.

Skill supervision	FWT ↑	NBT ↓	AUC ↑	Final SR ↑
CSG-generated labels	86.2	4.3	82.5	83.4
Verified reference labels	87.6	4.2	84.0	85.2

These results show that the automatic annotations support continual learning performance close to that obtained with verified supervision.

E.6 Feature-drift protocol

We assess old-task retention at the representation and behavior levels by comparing each selected LIBERO-Goal task's acquisition checkpoint with the final checkpoint.

Feature drift. Both checkpoints receive the same demonstration inputs, noise realizations, and denoising timestep. VLM drift measures the cosine distance between masked-mean visual-language prefix representations. Immediate-action drift measures the cosine distance between decoder hidden features at the first action position, before action projection. For horizon-action drift, we first average decoder hidden features over action positions for each checkpoint, then compute cosine distance. We average each metric over samples within a task and then equally over tasks. Fig. 4(a) normalizes each method's mean drift by the SCE mean for the same feature type.

Closed-loop divergence. Both checkpoints start from shared simulator-state anchors drawn from successful old-task rollouts. We measure the L2 distance between their simulator-state vectors after each executed action over a 40-step horizon. At each step, Fig. 4(b) averages distances over available paired anchors within each task and then equally over tasks; shading denotes pointwise 95% bootstrap confidence intervals over task means.

E.7 Adaptive balancing at skill transitions

DETE shifts the balance toward the Transition Expert Branch around skill boundaries, supporting coherent cross-skill transitions. Across LIBERO-Goal tasks, the Execution-branch fusion weight α decreases by 16.2% on average near transition boundaries (Fig. 6(a)). Representative transition windows also show an increase in the normalized contribution of the Transition Expert Branch (Fig. 6(b)). These patterns link adaptive fusion to the timing of skill changes: DETE emphasizes skill-specific execution within a skill and increases the relative contribution of transition modeling around skill boundaries.

E.8 Full-trajectory routing and old-expert stability

For a successful rollout of the previously learned task "open the top drawer and put the bowl inside," the final-stage checkpoint routes four contiguous phases through $E_0 \to E_1 \to E_2 \to E_0$. The mean

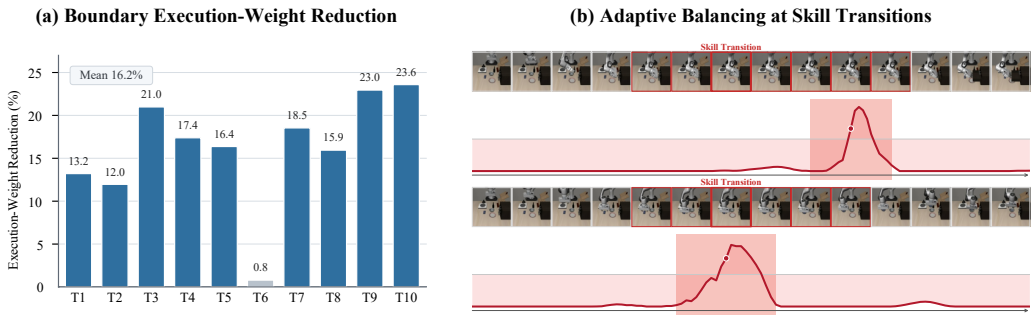


Figure 6: Adaptive balancing at skill transitions: (a) relative decrease in the Execution-branch fusion weight α around transition boundaries across LIBERO-Goal tasks; (b) increased normalized contribution of the Transition Expert Branch in representative transition windows.

dominant Execution probability in each phase is 95.7%, 96.4%, 96.2%, and 95.7%. The return to E_0 demonstrates reuse of the same expert at non-adjacent points of a complete trajectory. Across the three phase changes, Transition Experts reorganize their mixture weights; the largest measured increase is 13.6% at the final transition. An independently detected set of behavioral transition regions exhibits 3.74 times the routing redistribution observed in stable regions.

We further compare one old Execution Expert between its initial and final checkpoints using fixed old-task inputs. Top-1 skill routing is 100% accurate on the evaluated probe, and the forced-expert output features retain 98.39% cosine similarity. These results show that selective hard routing preserves the expert's functional role while allowing it to remain trainable through skill-relevant current and replay samples.

F Limitations and Future Work

SCE allocates one Execution Expert for each discovered skill. This design gives each reusable skill dedicated capacity and helps reduce interference among unrelated skills. However, it also means that the number of Execution Experts grows with the size of the skill base. In our experiments, the discovered skill bases are moderate in size ($K = 5$ for LIBERO-Goal, $K = 6$ for LIBERO-Long, $K = 2$ for LIBERO-Object, $K = 5$ for LIBERO-Spatial, and $K = 5$ for the real-world task stream), so the scalability of SCE under substantially larger and more diverse ECL streams remains an open question.

Future work should study more scalable expert-management strategies, such as sharing experts across semantically related skills, merging redundant experts after training, or organizing skills hierarchically. Another useful direction is to improve skill-base maintenance so that newly discovered skills are added only when they provide genuinely new reusable behavior, reducing unnecessary expert growth.

G Broader Impacts

This work studies embodied continual learning for robot manipulation. A positive impact is that more stable continual policies may reduce the need to retrain robots from scratch when new tasks are introduced, improving the adaptability and long-term reliability of embodied robots.

The same capability also carries deployment risks. A continually adapted robot may behave incorrectly under distribution shift, poor skill grounding, or unexpected object configurations. Such failures could damage objects or create unsafe interactions in physical environments. SCE is therefore intended as a research method for controlled evaluation, not as a complete safety mechanism for autonomous deployment. Practical deployment should include workspace constraints, action safety checks, human supervision when appropriate, and monitoring for behavioral drift after updates.

The experiments do not involve human subjects, personal data, or sensitive datasets. The real-world demonstrations are controlled robot-manipulation demonstrations collected in a tabletop environment.