Policy Compatible Skill Incremental Learning via Lazy Learning Interface

Daehee Lee[♠] **Dongsu Lee**[◇] **TaeYoon Kwack**[♠] **Wonje Choi**[♠] **Honguk Woo**[♠][*]
[♠]Sungkyunkwan University [◇]The University of Texas at Austin
dulgi7245@skku.edu, hwoo@skku.edu

Abstract

Skill Incremental Learning (SIL) is the process by which an embodied agent expands and refines its skill set over time by leveraging experience gained through interaction with its environment or by the integration of additional data. SIL facilitates efficient acquisition of hierarchical policies grounded in reusable skills for downstream tasks. However, as the skill repertoire evolves, it can disrupt compatibility with existing skill-based policies, limiting their reusability and generalization. In this work, we propose SIL-C, a novel framework that ensures skill-policy compatibility, allowing improvements in incrementally learned skills to enhance the performance of downstream policies without requiring policy re-training or structural adaptation. SIL-C employs a bilateral lazy learning-based mapping technique to dynamically align the subtask space referenced by policies with the skill space decoded into agent behaviors. This enables each subtask, derived from the policy's decomposition of a complex task, to be executed by selecting an appropriate skill based on trajectory distribution similarity. We evaluate SIL-C across diverse SIL scenarios and demonstrate that it maintains compatibility between evolving skills and downstream policies while ensuring efficiency throughout the learning process.
Source code: `https://github.com/L2dulgi/SIL-C`

1 Introduction

Lifelong embodied agents must continuously integrate novel knowledge from unceasing streams of data into their evolving skill library while simultaneously leveraging previously acquired skills [1–4]. *Skill Incremental Learning* (SIL) supports this process by enabling agents to expand and refine their skill sets over time through continual interaction with the environment or integration of new data. Accordingly, SIL often aims to facilitate the development of hierarchical policies grounded in reusable skills for downstream tasks [5–7].

Recent research explores diverse formulations of SIL, such as modular skill composition [8–11], continual adaptation [12–14], and hierarchical policy learning [15, 16] to support scalable embodied agents. Yet, a critical challenge remains underexplored; as skills evolve over time, maintaining compatibility with downstream policies that depend on those skills becomes increasingly difficult [17–20]. Without proper alignment, updated skills may invalidate previously learned policies, limiting their reusability and generalization [21–24].

Figure 1 illustrates a policy-compatible SIL scenario involving two types of skill-policy compatibility: (i) *Forward Skill Compatibility (FwSC)*, which ensures that a newly added skill can be effectively utilized during the training of future downstream policies, and (ii) *Backward Skill Compatibility (BwSC)*, which ensures that existing downstream policies can continue to use newly added or updated skills without requiring re-training, and, when applicable, benefit from improved policy performance.

[*]Corresponding author: Honguk Woo (hwoo@skku.edu)

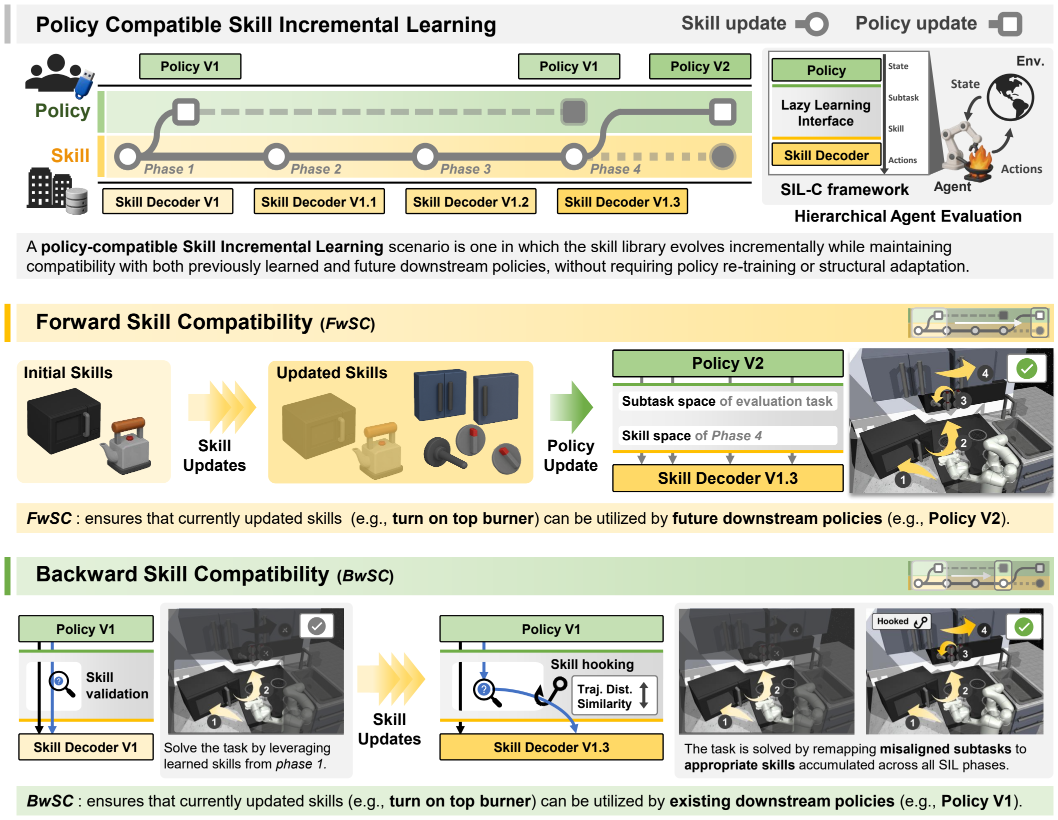

Policy Compatible Skill Incremental Learning
Skill update
Policy update
Policy
Skill
Policy V1
Policy V1
Policy V2
Phase 1
Phase 2
Phase 3
Phase 4
Skill Decoder V1
Skill Decoder V1.1
Skill Decoder V1.2
Skill Decoder V1.3
Policy
Lazy Learning Interface
Skill Decoder
State
Subtask
Skill
Actions
Env.
State
Actions
Agent
SIL-C framework
Hierarchical Agent Evaluation
A policy-compatible Skill Incremental Learning scenario is one in which the skill library evolves incrementally while maintaining compatibility with both previously learned and future downstream policies, without requiring policy re-training or structural adaptation.
Forward Skill Compatibility (FwSC)
Initial Skills
Skill Updates
Updated Skills
Policy Update
Policy V2
Subtask space of evaluation task
Skill space of Phase 4
Skill Decoder V1.3
FwSC : ensures that currently updated skills (e.g., turn on top burner) can be utilized by future downstream policies (e.g., Policy V2).
Backward Skill Compatibility (BwSC)
Policy V1
Skill validation
Skill Decoder V1
Solve the task by leveraging learned skills from phase 1.
Skill Updates
Policy V1
Skill hooking
Traj. Dist. Similarity
Skill Decoder V1.3
Hooked
The task is solved by remapping misaligned subtasks to appropriate skills accumulated across all SIL phases.
BwSC : ensures that currently updated skills (e.g., turn on top burner) can be utilized by existing downstream policies (e.g., Policy V1).

Figure 1: Overview of policy-compatible skill incremental learning and SIL-C framework

To address skill-policy compatibility challenges, we introduce SIL-C, a novel SIL framework with an interface layer, enabling hierarchical skill composition to seamlessly support downstream policy learning. For *FwSC*, inspired by append-only file systems, we design the framework to support non-destructive skill updates that mitigate forgetting and to introduce clear abstractions between task-specific policy learning and the task-agnostic evolving skill library, thereby enabling practical lifelong learning. For *BwSC*, we further design the interface within the framework to provide skill validation, which ensures that subtasks proposed by the high-level policy are contextually appropriate, and skill hooking, which intercepts misaligned subtasks and remaps them to the most suitable skill based on trajectory distribution similarity.

We implement both compatibility mechanisms through a lazy learning-based interface that maintains a consistent structure, enabling seamless integration with existing SIL approaches. To achieve *FwSC* and *BwSC* without requiring policy re-training or structural adaptation, we reformulate alignment between subtask space and skill space through trajectory distribution similarity as an instance-based classification problem with append-only prototype memories. By deferring decisions to inference time and offloading only the mapping function, this approach preserves existing training procedures while supporting the decoupled evolution of skills and policies.

With diverse SIL scenarios regarding compatibility, sample efficiency, and modularity, we evaluate SIL-C and other SIL baselines. SIL-C improves the normalized reward of downstream policies learned during the initial phase by an average of 18.6pp to 42.5pp across full scenarios, while the baselines only maintain or even degrade in performance. In particular, under the few-shot imitation setting in SIL, where each task is provided with only a single demonstration and 20% of its transitions, SIL-C achieves nearly double the overall performance of the baseline within the *BwSC* evaluation.

Our contributions are summarized as follows: (1) We propose SIL-C, a novel framework for SIL that maintains compatibility between evolving skills and downstream hierarchical policies. (2) We introduce a lazy learning-based mapping technique that aligns the subtask space from policies with the skill space based on trajectory distribution similarity. (3) We demonstrate the effectiveness of SIL-C across diverse SIL scenarios, showing that it maintains skill-policy compatibility while supporting efficient and scalable skill integration.

2 Related Work

Skill incremental learning. To achieve pre-trained behavior models for efficiently learning downstream tasks, prior work has explored unsupervised skill discovery [25, 26] and exploited task-agnostic data to build skill priors that can accelerate learning [27–30]. Contrastive objectives are also used to improve representation quality [31–34]. Several methods have been extended with new expert demonstrations to refine or expand priors for downstream adaptation [35–37]. Building on pre-trained models, recent work has investigated continual task adaptation in offline settings, aiming to handle streams of diverse tasks without full re-training [9, 38]. Beyond task-level adaptation, several approaches aim to incrementally acquire and organize skills at finer abstraction levels [39] by leveraging subgoal information [10, 11], often via hierarchical formulations [12, 13]. Yet, these SIL approaches typically assume synchronous updates between a skill library and downstream policies, limiting scalability when adding or removing skills.

Continual updates for pre-trained models. Expansion-based continual learning methods [40, 41] maintain task compatibility via explicit task identifiers, while recent adapter-based approaches [42–45] enable parameter-efficient updates. However, these typically rely on frozen backbones, limiting their ability to revise pre-trained knowledge [46]. Such issue arises in sequential decision-making [9, 11, 38], where pre-trained policies must adapt over time. Recently, a continual pre-training approach has been proposed to improve forward compatibility [47–49], but its integration into hierarchical settings remains underexplored. We explore this direction in the context of lifelong skill acquisition.

Lazy learning for behavior learning. Lazy learning, which defer learning until query time, reduces the cost of online adaptation and mitigate forgetting by leveraging instance-based retrieval and local updates [50]. They have been applied in behavior learning through prototype memories and nearest-neighbor matching [51–53], and in robotics applications, particularly for adapting to novel tasks from limited demonstrations [54–59]. While effective for adaptation, these methods typically do not support long-term skill reuse or accumulation without additional training. We address this by extending lazy learning into the SIL setting, reformulating trajectory-similarity-based skill validation and hooking as an instance-based classification mechanism that enables skill reuse and composition without policy re-training.

3 Problem Formulation

We formulate each task τ as a Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mu_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition probability of the environment, $\mathcal{R}$ is the reward function, μ_0 is the initial state distribution, and γ is the discount factor. The objective is to learn a policy $\pi(a|s;\theta)$ that maximizes the expected return:

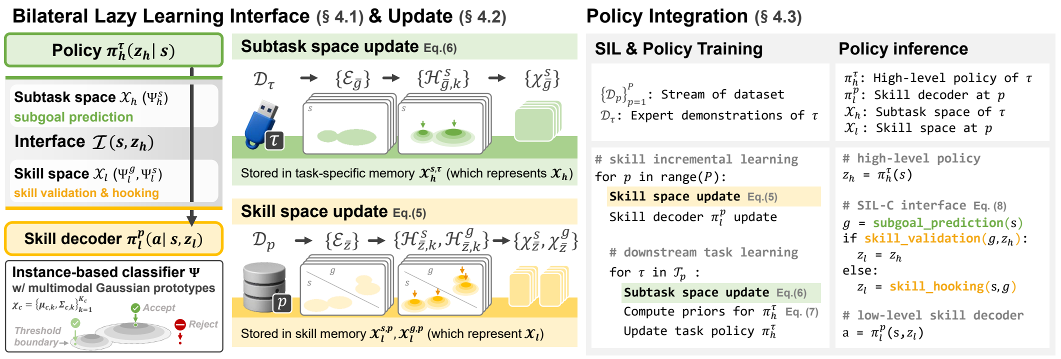
$$J(\theta) = \mathbb{E}_{\pi(a|s;\theta)}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)\right]. \tag{1}$$

To address long-horizon tasks, we adopt a hierarchical policy. A high-level policy $\pi_h(z_h|s;\theta_h)$ selects a subtask z_h, which is mapped to a skill $z_l = \psi(z_h)$ and executed by a low-level decoder $\pi_l(a|s, z_l;\theta_l)$. In our setup, the mapping ψ may be the identity map or a more general transformation, with both z_h and z_l lying in a shared latent space $\mathcal{Z}$. The overall objective is denoted as $J(\theta_l, \theta_h)$ [60].

In *Skill Incremental Learning* (SIL), we observe a stream of datasets $\{\mathcal{D}_p\}_{p=1}^{P}$, where each $\mathcal{D}_p$ contains demonstrations for updating the skill decoder π_l at phase p. The goal is to continually update the decoder with $\mathcal{D}_p$, while maintaining or improving performance across a set of evaluation tasks $\mathcal{T}_p$ presented at each phase. We assume access to task-level expert demonstrations $\mathcal{D}_\tau$ of τ for training the high-level policy π_h, enabling supervised learning and facilitating adaptation to evolving skill sets. The high-level policy for each task τ is updated independently of the phase and may be replaced or fixed. Our objective is to find the collections of high- and low-level optimal parameters(Θ_l^*, Θ_h^*) over all phases, that maximize performance across all observed tasks:

$$(\Theta_l^*, \Theta_h^*) = \underset{\Theta_l, \Theta_h}{\arg\max}\ \mathbb{E}_{\leq p}\left[\sum_{\tau \in \mathcal{T}_p} J_\tau(\theta_l^p, \theta_h^\tau)\right], \quad \Theta_l = \{\theta_l^p\}_{p=1}^{P}, \quad \Theta_h = \{\theta_h^\tau\}_{\tau \in \mathcal{T}}. \tag{2}$$

Here, θ_l^p and θ_h^τ denote the parameters of the skill decoder π_l^p at phase p and the high-level policy π_h^τ for task τ, respectively.


Bilateral Lazy Learning Interface (§ 4.1) & Update (§ 4.2)
Policy $\pi_h^\tau(z_h | s)$
Subtask space $\mathcal{X}_h$ (Ψ_h^s)
subgoal prediction
Interface $\mathcal{I}(s, z_h)$
Skill space $\mathcal{X}_l$ (Ψ_l^g, Ψ_l^s)
skill validation & hooking
Skill decoder $\pi_l^p(a | s, z_l)$
Instance-based classifier Ψ
w/ multimodal Gaussian prototypes
$\chi_c = \{\mu_{c,k}, \Sigma_{c,k}\}_{k=1}^{K_c}$
Accept
Reject
Threshold boundary
Subtask space update Eq.(6)
$\mathcal{D}_\tau \rightarrow \{\mathcal{E}_{\bar{g}}\} \rightarrow \{\mathcal{H}_{\bar{g},k}^s\} \rightarrow \{\chi_{\bar{g}}^s\}$
τ
Stored in task-specific memory $\mathcal{X}_h^{s,\tau}$ (which represents $\mathcal{X}_h$)
Skill space update Eq.(5)
$\mathcal{D}_p \rightarrow \{\mathcal{E}_{\bar{z}}\} \rightarrow \{\mathcal{H}_{\bar{z},k}^s, \mathcal{H}_{\bar{z},k}^g\} \rightarrow \{\chi_{\bar{z}}^s, \chi_{\bar{z}}^g\}$
p
Stored in skill memory $\mathcal{X}_l^{s,p}, \mathcal{X}_l^{g,p}$ (which represent $\mathcal{X}_l$)
Policy Integration (§ 4.3)
SIL & Policy Training
Policy inference

```
{D_p}_{p=1}^P: Stream of dataset
D_τ: Expert demonstrations of τ

# skill incremental learning
for p in range(P):
  Skill space update Eq.(5)
  Skill decoder π_l^p update

  # downstream task learning
  for τ in T_p :
    Subtask space update Eq.(6)
    Compute priors for π_h^τ Eq. (7)
    Update task policy π_h^τ
```

```
π_h^τ: High-level policy of τ
π_l^p: Skill decoder at p
X_h: Subtask space of τ
X_l : Skill space at p

# high-level policy
z_h = π_h^τ(s)

# SIL-C interface Eq. (8)
g = subgoal_prediction(s)
if skill_validation(g,z_h):
  z_l = z_h
else:
  z_l = skill_hooking(s,g)

# low-level skill decoder
a = π_l^p(s,z_l)
```

Figure 2: Overview of the SIL-C framework: components, updates, and integration

4 Our Framework

We introduce the SIL-C framework, a hierarchical architecture with a lazy learning interface designed to address skill-policy compatibility in SIL. It consists of a high-level policy π_h^τ, a low-level skill decoder π_l^p, and their respective learning algorithms A_h and A_l. The interface layer $\mathcal{I}$ connects the two levels by mapping each subtask z_h to a skill z_l. Both subtask and skill are drawn from the shared index set $\mathcal{Z}_p = \{1, \ldots, Z_p\}$, which expands over phases by appending new indices $\bar{\mathcal{Z}}_p = \{Z_{p-1}+1, \ldots, Z_p\}$. Given a state s, execution in SIL-C follows:

$$z_h \sim \pi_h^\tau(\cdot \mid s), \quad z_l = \mathcal{I}(s, z_h), \quad a \sim \pi_l^p(\cdot \mid s, z_l), \quad \text{where} \quad z_l, z_h \in \mathcal{Z}_p. \tag{3}$$

The core component of SIL-C is the bilateral lazy learning interface $\mathcal{I}$, which operates across two spaces: subtask space $\mathcal{X}_h$ and skill space $\mathcal{X}_l$. We implement this interface as a two-stage instance-based classifier that performs trajectory similarity matching between subtasks and skills. In our design, the subtask space $\mathcal{X}_h$ is represented by prototypes of subtask initial states used to predict the corresponding subgoal, while the skill space $\mathcal{X}_l$ is represented by prototypes that capture the distributions of initial states and subgoals for each skill. Together, these representations allow trajectory similarity to be compared across skills and subtasks. Building on this, interface operates in two stages: skill validation first checks whether the current subtask can achieve the task-specific subgoal predicted by $\mathcal{X}_h$ given the current state; if not, skill hooking remaps the subtask to the most feasible skill based on the current state. Figure 2 illustrates the components and their integration.

4.1 Bilateral Lazy Learning Interface

Bilateral modules ($\Psi_h^s, \Psi_l^g, \Psi_l^s$). To enable efficient trajectory distribution similarity matching under SIL, the interface $\mathcal{I}$ estimates similarity through bilateral lazy learning modules: the task-side module Ψ_h^s operating on the subtask space $\mathcal{X}_h$ and the skill-side modules Ψ_l^g, Ψ_l^s operating on the skill space $\mathcal{X}_l$. The two sides communicate via abstracted trajectory distributions represented by current state s and subgoal g pairs, where a subgoal represents a desired future state reachable from the current state s within m steps. To map subtask z_h to an executable skill z_l, the task-side module Ψ_h^s first predicts the target subgoal g from s. Then, the skill-side module Ψ_l^g assesses the direct executability of z_h toward g, for skill validation. If validation fails, skill-side modules Ψ_l^g, Ψ_l^s identify an appropriate skill via two-stage matching conditioned on (s, g), for skill hooking.

Instance-based classifier (Ψ). To implement lazy learning in the interface, we adopt an instance-based classifier for all modules Ψ. Each label $c \in \mathcal{C}$, representing a concrete prediction target (e.g., a subgoal g or a skill index z), is modeled by a multimodal Gaussian prototype $\chi_c = \{(\mu_{c,k}, \Sigma_{c,k})\}_{k=1}^{K_c}$, where K_c denotes the number of Gaussian components for label c, and $\Sigma_{c,k}$ is diagonal for computational efficiency. For a query instance x, the module computes distances to all prototypes:

$$d_c(x) = \min_{k \le K_c} \sqrt{(x - \mu_{c,k})^\top \Sigma_{c,k}^{-1} (x - \mu_{c,k})}. \tag{4}$$

This Mahalanobis distance-based multimodal approach effectively captures complex distributions for both accurate prediction and reliable validation. Specifically, it enables two operations: (i) classification via $\Psi(x; \mathcal{C}) = \arg\min_{c \in \mathcal{C}} d_c(x)$ for nearest prototype selection, and (ii) validation via $\Psi(x, c) = \mathbb{I}[d_c(x) \le \delta_c]$ for out-of-distribution detection, where δ_c is the square root of 99% chi-square quantile [61].

4.2 Interface Update

We parameterize the interface modules with append-only prototype memories. On the task-side, classifier Ψ_h^s relies on task-specific memory $\mathcal{X}_h^{s,\tau}$ to represent subtask space $\mathcal{X}_h$ for each task τ. On the skill side, classifiers Ψ_l^s and Ψ_l^g use task-agnostic shared memories $\mathcal{X}_l^{s,p}$ and $\mathcal{X}_l^{g,p}$ to represent skill space $\mathcal{X}_l$ at phase p. The subtask space $\mathcal{X}_h$ is updated during downstream policy learning, while the skill space $\mathcal{X}_l$ is updated during SIL.

Skill space update ($\mathcal{X}_l$). At SIL phase p, SIL-C updates $\mathcal{X}_l$ by generating skill prototypes from the streamed dataset $\mathcal{D}_p$. An unsupervised skill clustering algorithm [62, 60] segments $\mathcal{D}_p$ into distinct skill groups $\{\mathcal{E}_{\bar{z}}\}_{\bar{z}\in\bar{\mathcal{Z}}_p}$, where $\bar{\mathcal{Z}}_p$ is the index set of newly discovered skills at phase p. The number of discovered skills $\bar{\mathcal{Z}}_p$ can either be fixed or automatically determined by the clustering algorithm. For each group $\mathcal{E}_{\bar{z}}$, we apply K-means separately to the state and subgoal spaces to capture the multimodal distribution within the group. The number of sub-clusters $K_{\bar{z}}$ is selected either manually or automatically via silhouette score [63]. This process yields sub-clusters $\{\mathcal{H}_{\bar{z},k}^s, \mathcal{H}_{\bar{z},k}^g\}_{k=1}^{K_{\bar{z}}}$ for each skill $\bar{z}$. Then, we compute $\mu_{\bar{z},k}$ and $\Sigma_{\bar{z},k}$ to create prototype for each skill, denoted as $\chi_{\bar{z}}^s$ and $\chi_{\bar{z}}^g$, respectively, corresponding to skill label $\bar{z}$. The overall process follows:

$$\underbrace{\mathcal{D}_p}_{\text{skill dataset}} \xrightarrow[\text{segment}]{\text{Skill Clustering}} \underbrace{\{\mathcal{E}_{\bar{z}}\}_{\bar{z}\in\bar{\mathcal{Z}}_p}}_{\text{skill groups}} \xrightarrow[s,g]{K\text{-means}} \underbrace{\{\mathcal{H}_{\bar{z},k}^s, \mathcal{H}_{\bar{z},k}^g\}_{\bar{z}\in\bar{\mathcal{Z}}_p}^{k\in[1:K_{\bar{z}}]}}_{\text{sub-clusters}} \xrightarrow[s,g]{\mu_{\bar{z},k},\Sigma_{\bar{z},k}} \underbrace{\{\chi_{\bar{z}}^s\,, \chi_{\bar{z}}^g\}_{\bar{z}\in\bar{\mathcal{Z}}_p}}_{\text{skill prototypes}}. \tag{5}$$

The resulting skill prototypes are stored in skill memory $\mathcal{X}_l^{s,p}$ and $\mathcal{X}_l^{g,p}$.

Subtask space update ($\mathcal{X}_h$). At downstream policy learning for task τ, SIL-C constructs $\mathcal{X}_h$ by generating subtask prototypes from the most recent expert demonstrations $\mathcal{D}_\tau$, to the discretized subgoal groups $\mathcal{G}_\tau$. To efficiently represent demonstrations, we apply the same skill clustering algorithm [60] to derive $\mathcal{G}_\tau$, followed by the same sub-clustering procedure as in the skill space update:

$$\underbrace{\mathcal{D}_\tau}_{\text{expert dataset}} \xrightarrow[\text{segment}]{\text{Subtask Clustering}} \underbrace{\{\mathcal{E}_{\bar{g}}\}_{\bar{g}\in\mathcal{G}_\tau}}_{\text{subtask groups}} \xrightarrow[s]{K\text{-means}} \underbrace{\{\mathcal{H}_{\bar{g},k}^s\}_{\bar{g}\in\mathcal{G}_\tau}^{k\in[1:K_{\bar{g}}]}}_{\text{sub-clusters}} \xrightarrow[s]{\mu_{\bar{g},k},\Sigma_{\bar{g},k}} \underbrace{\{\chi_{\bar{g}}^s\}_{\bar{g}\in\mathcal{G}_\tau}}_{\text{subtask prototypes}} \quad . \tag{6}$$

After prototype construction, subtask prototypes are stored in the task-specific memory $\mathcal{X}_h^{s,\tau}$.

4.3 Policy Integration

Policy learning on the interface. We utilize an energy-based prior [64] to guide the learning of the high-level policy A_h. This prior is induced by the skill decoder: given a state s, the decoder evaluates all candidate skill pairs $\{(s, z_l)\}_{z_l\in\mathcal{Z}_p}$ and assigns the subtask label whose decoded action best matches the expert action a^*. Specifically, we compute the subtask label as:

$$z_h = \underset{z_l\in\mathcal{Z}_p}{\arg\min} \|\hat{a} - a^*\|^2 \tag{7}$$

where $\hat{a} \sim \pi_l(\cdot \mid s, z_l)$ denotes the decoded action from the skill decoder using skill z_l. The decoder is continually updated by the skill incremental learning algorithm A_l.

Policy inference via the interface. The policy inference process follows the sequence defined in Eq. (3). Given a sampled subtask $z_h \sim \pi_h^\tau(\cdot \mid s)$ from the high-level policy, the interface first predicts the subgoal $g = \Psi_h^s(s; \mathcal{G}_\tau)$ using the task-side module, conditioned on the task-specific memory $\mathcal{X}_h^{s,\tau}$. Using this inferred subgoal g, the interface then performs skill validation by evaluating the distance of subgoal-subtask pair (g, z_h) by skill-side module $\Psi_l^g(g, z_h)$. If the skill validation passes, the interface returns $z_l = z_h$. Otherwise, it initiates skill hooking, seeking an alternative skill better aligned with the inferred subgoal g. Formally, the decision rule for interface $\mathcal{I}$ follows:

$$\mathcal{I}(s, z_h) = \begin{cases} z_h, & \Psi_l^g(g, z_h) = 1 \\ \Psi_l^s(s; \mathcal{Z}'), & \Psi_l^g(g, z_h) = 0 \end{cases} \quad \text{, where} \quad g = \Psi_h^s(s; \mathcal{G}_\tau). \tag{8}$$

Here, candidate skill set $\mathcal{Z}' = \{z' \in \mathcal{Z}_p | \Psi_l^g(g, z') = 1\} \cup \{z_h\}$, where z_h, originally inferred subtask from the high-level policy, is included as a default fallback when no valid candidate skill exists.

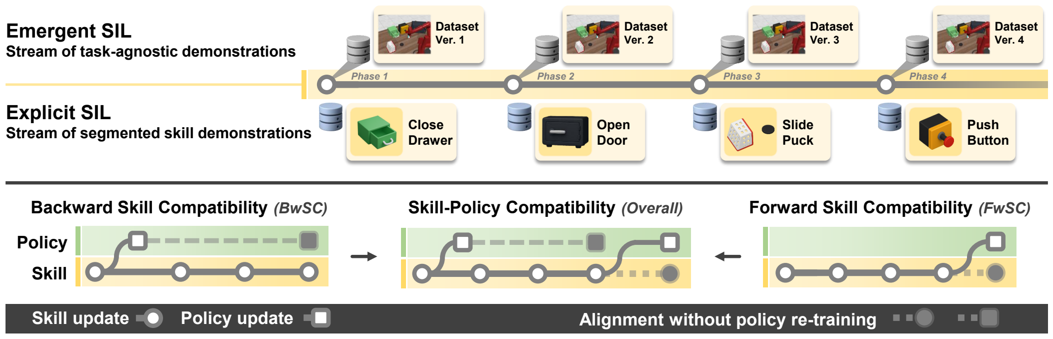

Emergent SIL
Stream of task-agnostic demonstrations
Dataset Ver. 1
Dataset Ver. 2
Dataset Ver. 3
Dataset Ver. 4
Phase 1
Phase 2
Phase 3
Phase 4
Explicit SIL
Stream of segmented skill demonstrations
Close Drawer
Open Door
Slide Puck
Push Button
Backward Skill Compatibility (BwSC)
Skill-Policy Compatibility (Overall)
Forward Skill Compatibility (FwSC)
Policy
Skill
Skill update
Policy update
Alignment without policy re-training

Figure 3: (*Top*) SIL scenario types and (*Bottom*) evaluation groups

5 Experiments

5.1 Experiment Settings

Environments and SIL scenarios. To evaluate skill-policy compatibility, we construct various SIL scenarios using two simulation environments: Franka Kitchen [65, 66] and Meta-World [67, 11]. In our experiments, we have four SIL phases where we add new skills by training the skill decoder, and train 24 downstream task policies using the updated skills for each phase.

As illustrated in the top of Figure 3, we define two scenario types to evaluate SIL under different levels of skill supervision. The *Emergent SIL* setting follows skill-based pretraining approaches [28, 27], introduces only task-agnostic demonstrations, reflecting an unsupervised SIL setup. In contrast, the *Explicit SIL* setting adopts the setup from prior SIL works [12, 68] and uses demonstrations that exhibit clearly chunked behaviors with predefined semantics.

As shown in the bottom of Figure 3, we categorize each scenario into three evaluation groups based on the combinations of skill decoders and policies used for task evaluation. The *BwSC group* pairs the skill decoder from each phase with the policy initially trained in the first phase. The *FwSC group* pairs the skill decoder from each phase with its synced policy trained using that decoder. The *Overall group* aggregates both *BwSC* and *FwSC* groups to measure comprehensive compatibility.

Baselines. We evaluate three baseline groups. **Type I** combines skill-based pre-training with continual learning strategies for SIL, adopting a simple skill appending scheme. We extend BUDS [62, 10] and PTGM [60], which discretize the skill space during pre-training. As continual learning methods, we apply fine-tuning (FT), experience replay (ER) [69], and adapter append (AA) [38], which trains an adapter after the initial phase and stores it for use in subsequent phases. **Type II** includes SIL approaches that use semantic goals as skill labels, relying on pre-defined supervision. They include prototype-based skill retrieval [11] and temporal replay with model expansion [12]. **Type III** corresponds to SIL-C, which uses the Type I configuration as its base architecture. We also report joint-training performance, obtained by training a single decoder on the union of all skill datasets.

All baselines adopt the same goal-conditioned skill decoder architecture [60], which conditions on subgoals hashed by skills, and use a diffusion-based model [70, 71] for decoding. For policy learning, they follow the same training objective, behavior cloning with a prior, as defined in Eq. (7).

Metrics. We measure skill-policy compatibility by adapting continual learning metrics from lifelong robot learning [4, 11, 38]. All task evaluation results are normalized to 100% based on the maximum reward defined in each environment. Forward Transfer (FWT) measures the performance of newly trained policies after skill updates, quantifying how updated skills facilitate learning new tasks. We report FWT at the *Initial* and *Final* phases of SIL. Backward Transfer (BWT) measures the change in performance from initial evaluation, assessing how skill updates affect previously evaluated tasks. Area Under the Curve (AUC) averages all phase performances within a scenario. All metrics are measured per task and reported as averages across tasks for each evaluation group in each scenario.

For comprehensive experiment settings including implementation details, hyperparameters, and additional configurations, please refer to the Appendix.

Table 1: Performance evaluation of skill-policy compatibility in two SIL scenarios of the Kitchen environment with four seeds. Each row corresponds to a baseline, categorized by the skill interface configuration, hierarchical agent structure, and the SIL algorithm used for skill decoder updates. The symbol * denotes methods that require pre-defined semantic skill labels for both policy and decoder training. The left side reports *BwSC* results, the right side reports *FwSC* results, and the center shows the overall performance considering both. Best results are shown in **bold**. In this table, SIL-C uses the Type I (PTGM + AA) setting as its base architecture.

Kitchen : *Emergent SIL*										
Baselines			*Initial*	*BwSC* (*Initial*)		Overall performance		*FwSC* (*Synced*)		*Final*
Type	Skill Interface	SIL Algo.	FWT (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	FWT (%)
I	Skill Segments [62, 10] (BUDS)	FT	23.5±4.9	-16.4±5.8	11.2±1.2	-6.1±4.9	18.3±1.2	4.3±4.3	26.7±2.3	18.7±1.2
		ER	23.5±3.8	-10.8±8.1	15.5±2.3	3.6±7.0	26.6±2.8	17.9±6.5	37.0±3.0	44.5±7.0
		AA	21.9±4.1	1.3±4.4	22.9±4.0	15.7±3.5	35.3±2.9	30.1±3.9	44.4±2.6	57.1±7.5
	Subgoal Bins [60] (PTGM)	FT	45.0±2.0	-36.9±1.5	17.3±0.9	-24.3±1.6	24.1±1.0	-11.7±2.1	36.2±1.6	24.7±2.0
		ER	46.8±2.5	-19.7±4.4	32.1±2.9	-5.0±2.8	42.5±1.4	9.7±3.3	54.0±1.8	57.7±3.6
		AA	45.0±3.1	2.6±0.9	46.9±3.7	15.4±1.7	58.2±1.7	28.1±4.1	66.1±0.6	83.6±3.1
II	Skill-prototypes [11]*	AA	55.1±1.8	0.5±3.1	55.5±2.7	0.5±2.8	55.7±2.7	0.9±2.2	55.8±2.4	56.9±4.9
	Instructions [12]*	ER	54.0±2.4	18.5±4.0	**67.8**±1.1	19.1±2.7	70.3±1.0	19.7±2.0	68.7±1.9	77.4±2.8
III	SIL-C (w/ PTGM)	AA	52.9±2.9	18.6±1.2	66.8±2.6	22.0±2.4	**71.8**±1.3	25.4±4.0	**71.9**±0.9	87.2±3.2
-	Subgoal Bins [60]	Joint	-	-	-	-	-	-	-	86.9±2.2
Kitchen : *Explicit SIL*										
Baselines			*Initial*	*BwSC* (*Initial*)		Overall performance		*FwSC* (*Synced*)		*Final*
Type	Skill Interface	SIL Algo.	FWT (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	FWT (%)
I	Skill Segments [62, 10] (BUDS)	FT	14.6±0.0	-13.6±1.0	4.4±0.8	-9.5±0.6	6.5±0.5	-5.4±0.6	10.5±0.4	3.0±1.9
		ER	14.6±0.0	0.2±0.5	14.7±0.4	14.1±1.3	26.7±1.1	28.0±2.2	35.6±1.7	37.0±2.8
		AA	14.6±0.0	0.0±0.0	14.6±0.0	20.9±1.1	32.5±1.0	41.9±2.3	46.0±1.7	62.8±3.0
	Subgoal Bins [60] (PTGM)	FT	14.6±0.0	-14.4±0.2	3.8±0.2	-8.9±1.0	7.0±0.9	-1.7±3.9	12.1±1.5	1.7±2.1
		ER	14.6±0.3	-1.5±1.1	13.5±0.8	17.2±1.2	29.3±1.1	35.9±1.7	41.5±1.5	52.0±7.5
		AA	14.5±0.2	0.0±0.1	14.6±0.4	25.8±1.0	36.7±0.8	51.6±1.9	**53.3**±1.4	79.8±2.2
II	Skill-prototypes [11]*	AA	14.6±0.0	0.0±0.0	14.6±0.0	18.4±11.3	33.6±3.5	44.3±8.1	47.8±6.1	75.0±12.7
	Instructions [12]*	ER	14.6±0.0	-0.6±2.7	14.1±2.1	19.1±3.4	30.9±2.9	38.7±4.4	43.6±3.3	60.2±8.8
III	SIL-C (w/ PTGM)	AA	14.6±0.0	42.5±3.0	**46.5**±2.3	46.2±2.5	**54.1**±2.1	49.8±2.4	51.9±1.8	80.6±6.3
-	Subgoal Bins [60]	Joint	-	-	-	-	-	-	-	86.9±2.2

5.2 Skill-Policy Compatibility : Backward and Forward

Table 1 presents the skill-policy compatibility evaluation for the Franka Kitchen environment, covering both *Emergent* and *Explicit* skill incremental scenarios. It jointly reports *BwSC* and *FwSC*, and the overall performance shows that SIL-C achieves the highest AUC among all baselines.

Backward Skill Compatibility (*BwSC*). The left side of Table 1 reports the performance of the *Initial* phase policy after subsequent skill updates, evaluated across all phases. Compared to Type I baselines, SIL-C achieves a higher initial performance (*Initial* FWT) and closely matches the performance of methods with predefined skill labels. This suggests that SIL-C can robustly replace lower-confidence skills via skill validation and hooking. Furthermore, SIL-C matches the BWT and AUC scores of Type II methods. However, this is partly due to the nature of the *Emergent* scenario in Franka Kitchen, where future skills are already available and fully observable from the start, creating a near-oracle setting. In *Explicit* scenarios with incrementally revealed skill labels, SIL-C consistently improves performance, whereas most Type II methods show near-zero BWT, merely preserving initial performance without adaptation. Meanwhile, Type I baselines with limited replay suffer from skill forgetting, with only the *AA* variant maintaining backward compatibility but failing to improve it.

Forward Skill Compatibility (*FwSC*). The right side of Table 1 reports performance when policies are retrained at each phase to align with updated skills. SIL-C and PTGM with append-only skill learning achieve comparable final performance (*Final* FWT) to joint training, highlighting effective utilization of accumulated skills. It also achieves the highest AUC, indicating strong forward compatibility. Among Type I methods, ER and AA yield positive BWT, suggesting that newly acquired skills contribute to subsequent policy learning, although less effectively than SIL-C.

In Appendix C, we provide additional overviews and extend the experiments with diverse configurations of SIL algorithms to further motivate our design choices, and present experiments with varying SIL phase orderings, showing that SIL-C consistently maintains skill-policy compatibility.

Table 2: Few-shot imitation results on Kitchen *Emergent* SIL. **Shots** denotes expert demonstrations per task for high-level policy training. **Ratio** indicates the proportion of transitions uniformly sampled from each demonstration. All baselines use SIL algorithm AA for skill decoder.

Kitchen : *Emergent SIL*											
Baselines				*Initial*	*BwSC* (*Initial*)		Overall performance		*FwSC* (*Synced*)		*Final*
Type	Skill Interface	Shots	Ratio	FWT (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	FWT (%)
I	Subgoal Bins [60] (PTGM)	5	100%	40.5±0.9	0.4±2.4	40.8±2.1	10.0±1.4	49.1±2.0	19.6±2.0	55.3±2.4	67.6±6.0
		3	100%	34.7±2.1	0.7±4.9	35.1±2.5	5.9±4.8	38.6±2.0	11.0±5.6	40.2±1.0	45.7±3.6
		1	100%	26.5±1.9	1.7±2.5	27.7±2.2	6.1±2.4	31.7±0.9	10.5±3.4	34.3±0.9	37.8±1.9
		1	50%	28.6±5.7	-0.8±4.4	27.9±3.2	1.0±5.1	29.5±1.8	2.9±6.0	30.8±1.2	36.5±4.0
		1	20%	25.7±3.5	-2.9±2.0	23.5±2.2	1.5±4.8	27.0±1.2	5.9±7.6	30.1±2.5	30.5±4.4
III	SIL-C (w/ PTGM)	5	100%	47.4±3.2	11.3±2.4	55.9±4.6	17.5±1.2	62.4±2.7	23.6±4.2	65.1±2.3	75.8±4.4
		3	100%	44.4±1.8	11.2±2.5	52.8±2.9	14.9±2.2	57.2±1.8	18.6±3.3	58.4±1.2	68.5±4.0
		1	100%	43.1±5.4	11.9±2.8	52.0±5.3	15.7±3.1	56.5±3.7	19.4±4.8	57.6±2.9	65.7±3.6
		1	50%	41.6±4.8	11.7±2.3	50.4±4.7	14.9±2.6	54.3±2.7	18.0±4.7	55.1±1.6	67.2±3.5
		1	20%	37.3±6.4	12.0±3.3	46.4±5.7	14.4±5.6	49.7±2.6	16.8±9.9	49.9±3.0	58.5±8.1

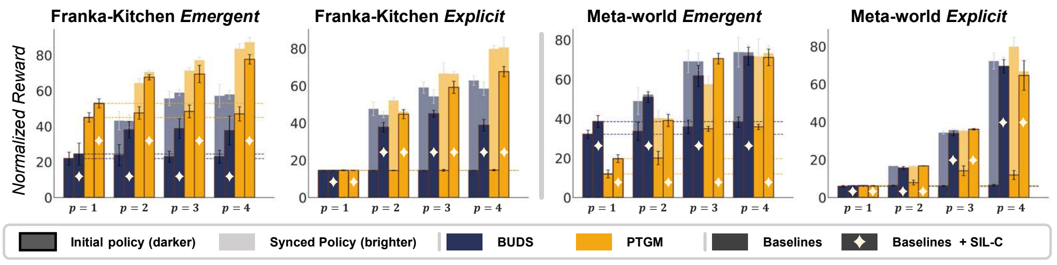


Figure 4: Results on Kitchen and Meta-World tasks under *Emergent* and *Explicit* skill incremental scenarios. Each group of bars represents a skill update phase (x-axis), and the y-axis shows the scaled reward. Darker bars represent evaluation with the initial policies. Lighter bars show performance after re-training with updated skills. Diamonds ($\diamondsuit$) indicate the application of SIL-C to each baseline. The skill decoder uses the AA strategy for Kitchen and ER for Meta-World.

5.3 Sample Efficiency: Downstream Few-shot Imitation Learning

Effect of number of demonstrations. Table 2 shows that across all demonstration budgets, SIL-C outperforms the baseline, highlighting the benefit of skill validation and hooking via lazy learning under limited supervision. With 5 demonstrations per task (5-shots), SIL-C improves overall performance AUC from 49.1% to 62.4% and *Final* FWT from 67.6% to 75.8%, indicating utility even with ample supervision. When using only 1 demonstration per task (1-shot), the gains become more significant, with AUC increasing from 31.7% to 56.5% and *Final* FWT from 37.8% to 65.7%.

Effect of transition sampling ratio. Reducing the number of transitions extracted from demonstrations (from 100% to 20%) has only a moderate effect on SIL-C, which still maintains a positive BWT and achieves an AUC above 46% on *BwSC*. In contrast, the baseline degrades substantially, with AUC dropping below 24%. This indicates that the skill-conditioned interface provides robustness to sparse supervision by substituting missing transitions with reusable behaviors. These results suggest that the interface enables strong generalization by leveraging prior skills in low-data regimes.

5.4 Modularity: Under Varying Design Choices for Hierarchical Architecture

Figure 4 shows that across all scenarios, SIL-C consistently improves *Initial Policy* performance by enhancing *BwSC*, regardless of environment, skill clustering strategy, or SIL algorithm for low-level decoder. In the Franka Kitchen environment, as shown in Table 1, SIL-C enables both BUDS and PTGM with AA to make more effective use of their accumulated skills, leading to more efficient reuse. This effect is clearly reflected in the visualized performance. In the Meta-World environment, unlike in Kitchen, both BUDS and PTGM use ER to train a shared skill decoder. While this promotes generalization and enables reuse of new skills without policy re-training, the actual gains remain limited. When SIL-C is applied, we observe a clear improvement in evaluation with the initial policy, suggesting that skill validation and hooking are effective in filtering unreliable skills. Although SIL-C relies less on broadly generalized skills from ER, it tends to reuse verified skills more precisely, which may contribute to safer and more stable policy execution.

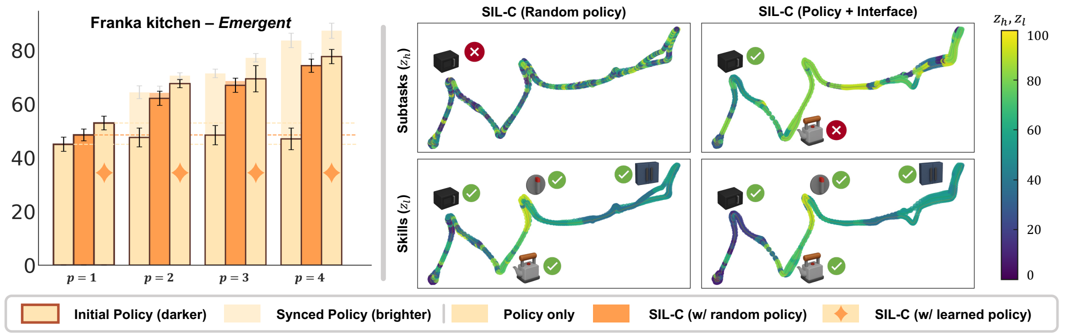


Figure 5: Ablation of the SIL-C lazy learning interface in the Kitchen *Emergent SIL* scenario using PTGM with AA configuration. (*Left*) Per-phase performance across three settings. (*Right*) Evaluation trajectories example illustrating selected subtasks and corresponding skills used to solve the task: `open microwave` $\rightarrow$ `move kettle` $\rightarrow$ `turn on top burner` $\rightarrow$ `open hinge cabinet`.

Table 3: Performance under varying input noise levels in Kitchen *Emergent SIL* scenario. All baselines use the same SIL algorithm AA for the decoder.

Kitchen : *Emergent SIL*										
Baselines			*Initial*	*BwSC* (*Initial*)		Overall performance		*FwSC* (*Synced*)		*Final*
Type	Skill Interface	Noise	FWT (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	FWT (%)
I	Subgoal Bins [60] (PTGM)	×1	45.0±3.1	2.6±0.9	46.9±3.7	15.4±1.7	58.2±1.7	28.1±4.1	66.1±0.6	83.6±3.1
		×2	44.4±4.5	0.8±2.0	45.0±3.4	10.7±2.7	53.5±2.2	20.5±3.9	59.7±1.9	68.6±3.5
		×3	35.8±5.3	0.5±3.7	36.1±2.9	3.2±5.6	38.5±1.5	5.9±7.7	40.2±1.7	40.5±4.6
		×5	19.7±1.7	-1.2±3.4	18.8±1.7	-1.8±2.4	18.2±0.9	-2.4±2.5	17.9±1.1	15.9±1.6
III	SIL-C (w/ PTGM)	×1	52.9±2.9	18.6±1.2	66.8±2.6	22.0±2.4	71.8±1.3	25.4±4.0	71.9±0.9	87.2±3.2
		×2	49.7±0.6	11.3±1.8	58.2±1.5	16.7±1.1	64.0±1.3	22.1±1.9	66.3±1.8	76.4±2.6
		×3	35.6±2.2	13.6±4.2	45.8±3.1	15.4±2.9	48.8±1.5	17.3±3.4	48.5±1.0	52.9±3.9
		×5	19.6±2.3	4.3±1.3	22.8±2.4	3.8±2.4	22.9±1.4	3.3±4.1	22.1±1.4	23.1±4.1

6 Ablation and Analysis

6.1 Lazy Learning Interface Ablation

Figure 5 ablates lazy learning interface and policy integration contributions across three settings: policy-only, SIL-C with a random policy, and SIL-C with a learned policy. While the policy-only configuration cannot leverage newly added skills without retraining. In contrast, the SIL-C w/ random policy shows that skill validation and hooking mechanisms alone offer minimal decision-making capability via trajectory distribution similarity matching, but still underperform due to limited generalization ability compared to using a learned high-level policy. The SIL-C w/ learned policy further improves performance by using skill validation with out-of-distribution detection to correct subtask-skill mismatches, which is crucial for error-sensitive long-horizon tasks. Evaluation trajectories show that while the random policy fails immediately from inconsistent subtask selection and the policy-only configuration struggles with the not-yet-acquired skill `move kettle`, SIL-C completes tasks through the interface remapping subtasks to utilize newly added skills.

6.2 Robustness

Table 3 evaluates robustness in the Kitchen environment by scaling input noise during evaluation, where the noise parameter [65] is increased from $\times 1$ to $\times 5$. SIL-C consistently outperforms PTGM+AA across all noise levels. Notably, under $\times 5$ noise, SIL-C maintains a positive BWT of 4.3%. The performance gap further increases with larger skill library sizes. For example, under $\times 3$ noise, the FWT difference grows from a marginal -0.2% *Initial* FWT to 12.4% *Final* FWT at convergence with 80 skills, where SIL-C achieves 52.9% compared to 40.5% for PTGM+AA.

This robustness stems from architectural differences in handling noisy observations. When noise corrupts high-level policy outputs, the bilateral interface in SIL-C performs skill validation to detect out-of-distribution subtasks and skill hooking to remap them based on trajectory similarity. In contrast, the static skill mapping in PTGM+AA cannot adapt to distributional shifts, leading to error accumulation in long-horizon tasks. The widening performance gap suggests that trajectory-based matching becomes increasingly effective as the skill repertoire expands.

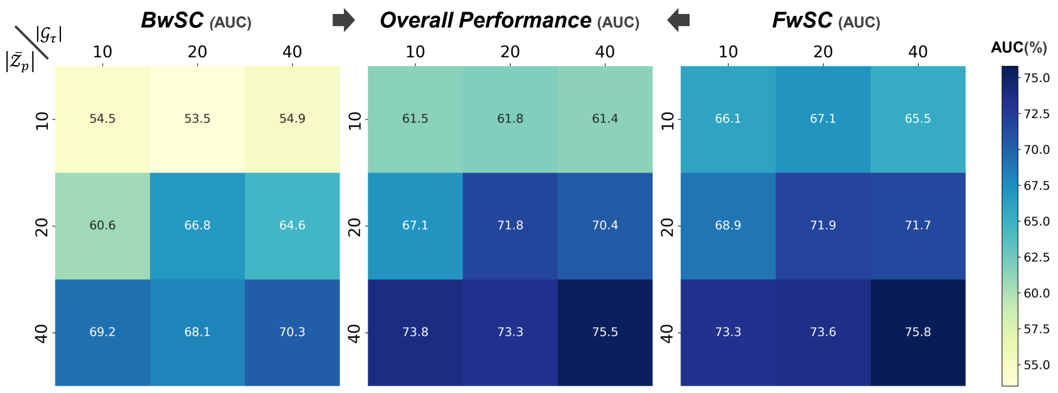


Figure 6: Analysis with varying $|\bar{\mathcal{Z}}_p|$ and $|\mathcal{G}_\tau|$ of SIL-C (w/ PTGM, *AA*) in Kitchen *Emergent SIL* scenario. Rows represent different values of $|\bar{\mathcal{Z}}_p|$ while columns represent different values of $|\mathcal{G}_\tau|$. AUC results shown for (*Left*) *BwSC*, (*Center*) Overall performance, and (*Right*) *FwSC*.

6.3 Skill and Subtask Space Resolution

Figure 6 examines how the resolution of skill and subtask spaces influences *FwSC* and *BwSC*. We vary the number of skill groups per phase ($|\bar{\mathcal{Z}}_p|$) and subtask groups per task ($|\mathcal{G}_\tau|$) in SIL-C (with PTGM and AA). Starting from the default setting in Table 1 ($|\bar{\mathcal{Z}}_p| = 20$, $|\mathcal{G}_\tau| = 20$), we test doubled and halved values to assess sensitivity to space resolution, which is determined by the granularity and quality of unsupervised clustering. Full results are reported in Appendix D.1.

Skill space resolution ($|\bar{\mathcal{Z}}_p|$). Increasing $|\bar{\mathcal{Z}}_p|$ from 10 to 40 raises overall AUC from 61.5% to 75.5% when $|\mathcal{G}_\tau| = 40$. This improvement is consistent across all subtask settings, with $|\bar{\mathcal{Z}}_p| = 40$ yielding the best performance in both *BwSC* (70.3%) and *FwSC* (75.8%). Higher skill space resolution also reduces dependence on subtask space resolution. With $|\bar{\mathcal{Z}}_p| = 40$, *BwSC* remains stable across subtask configurations (69.2%–70.3%), indicating that sufficient skill diversity compensates for coarse subtask space.

Subtask space resolution ($|\mathcal{G}_\tau|$). The effect of subtask space resolution depends on skill space resolution. With coarse skill space ($|\bar{\mathcal{Z}}_p| = 10$), increasing $|\mathcal{G}_\tau|$ from 10 to 40 changes performance only marginally (61.5% to 61.4% in *Overall Performance*). In contrast, with finer skill space ($|\bar{\mathcal{Z}}_p| = 40$), the same increase yields gains (73.8% to 75.5%). This shows that sufficient skill diversity is necessary for benefiting from finer subtask space, as coarse skill space limits the effect of additional subtask prototypes.

7 Conclusion

We present SIL-C, a framework for policy-compatible skill incremental learning (SIL) that introduces a bilateral lazy learning interface to preserve *FwSC* and *BwSC* without requiring full re-training or structural adaptation of downstream policies. Empirical results demonstrate that, by ensuring skill-policy compatibility, SIL-C consistently improves performance, efficiency, and modularity.

Future Directions. While our approach performs well in the simulation-based environments with well-defined state representations, future work may explore settings with more diverse or noisy skill distributions, where unsupervised clustering alone becomes less effective. Enhancing robustness in such conditions and leveraging minimal goal information for exploration can improve sample efficiency. Additionally, monitoring skill reliability and distribution shifts during deployment may help enforce expected behaviors and detect anomalies, contributing to safer execution.

Impact Statement. Our framework aims to support large-scale deployment in practical settings where new skills must be integrated without disrupting existing behaviors. By maintaining policy compatibility and enabling decentralized skill sharing, SIL-C provides a foundation for scalable robotic systems that can evolve over time. This line of research has potential to assist the development of adaptive, modular agents in applied domains.

References

[1] David Abel, Andre Barreto, Benjamin Van Roy, Doina Precup, Hado van Hasselt, and Satinder Singh. A definition of continual reinforcement learning. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, 2023.

[2] Edward Hughes, Michael D Dennis, Jack Parker-Holder, Feryal Behbahani, Aditi Mavalankar, Yuge Shi, Tom Schaul, and Tim Rocktäschel. Position: Open-endedness is essential for artificial superhuman intelligence. In *Forty-first International Conference on Machine Learning (ICML)*, 2024.

[3] Yuan Meng, Zhenshan Bing, Xiangtong Yao, Kejia Chen, Kai Huang, Yang Gao, Fuchun Sun, and Alois Knoll. Preserving and combining knowledge in robotic lifelong reinforcement learning. *Nature Machine Intelligence*, pages 1–14, 2025.

[4] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, qiang liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track (NeurIPS)*, 2023.

[5] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 1999.

[6] Alexander Li, Carlos Florensa, Ignasi Clavera, and Pieter Abbeel. Sub-policy adaptation for hierarchical reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2020.

[7] Eleftherios Triantafyllidis, Fernando Acero, Zhaocheng Liu, and Zhibin Li. Hybrid hierarchical learning for solving complex sequential tasks using the robotic manipulation network roman. *Nature Machine Intelligence*, 2023.

[8] Mikel Malagon, Josu Ceberio, and Jose A Lozano. Self-composing policies for scalable continual reinforcement learning. In *Forty-first International Conference on Machine Learning (ICML)*, 2024.

[9] Thomas Schmied, Markus Hofmarcher, Fabian Paischer, Razvan Pascanu, and Sepp Hochreiter. Learning to modulate pre-trained models in rl. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[10] 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)*, 2024.

[11] Daehee Lee, Minjong Yoo, Woo Kyung Kim, Wonje Choi, and Honguk Woo. Incremental learning of retrievable skills for efficient continual task adaptation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[12] Zexin Zheng, Jia-Feng Cai, Xiao-Ming Wu, Yi-Lin Wei, Yu-Ming Tang, and Wei-Shi Zheng. imanip: Skill-incremental learning for robotic manipulation. *arXiv preprint arXiv:2503.07087*, 2025.

[13] Jingkai Xu and Xiangli Nie. Speci: Skill prompts based hierarchical continual imitation learning for robot manipulation. *arXiv preprint arXiv:2504.15561*, 2025.

[14] Yixiao Wang, Yifei Zhang, Mingxiao Huo, Thomas Tian, Xiang Zhang, Yichen Xie, Chenfeng Xu, Pengliang Ji, Wei Zhan, Mingyu Ding, and Masayoshi Tomizuka. Sparse diffusion policy: A sparse, reusable, and flexible policy for robot learning. In *Proceedings of The 8th Conference on Robot Learning (CoRL)*, 2025.

[15] George Konidaris and Andrew Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in neural information processing systems (NeurIPS)*, 2009.

[16] Nur Muhammad Mahi Shafiullah and Lerrel Pinto. One after another: Learning incremental skills for a changing world. In *International Conference on Learning Representations (ICLR)*, 2022.

[17] Nico Gürtler, Dieter Büchler, and Georg Martius. Hierarchical reinforcement learning with timed subgoals. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021.

[18] Hao Ju, Rongshun Juan, Randy Gomez, Keisuke Nakamura, and Guangliang Li. Transferring policy of deep reinforcement learning from simulation to reality for robotics. *Nature Machine Intelligence*, 2022.

[19] Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 2022.

[20] Hongjoon Ahn, Jinu Hyeon, Youngmin Oh, Bosun Hwang, and Taesup Moon. Prevalence of negative transfer in continual reinforcement learning: Analyses and a simple baseline. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[21] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems (NeurIPS)*, 2016.

[22] Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. *Advances in neural information processing systems (NeurIPS)*, 2018.

[23] Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. Individualized controlled continuous communication model for multiagent cooperative and competitive tasks. In *International Conference on Learning Representations (ICLR)*, 2019.

[24] Ziluo Ding, Tiejun Huang, and Zongqing Lu. Learning individually inferred communication for multi-agent cooperation. *Advances in neural information processing systems (NeurIPS)*, 33:22069–22079, 2020.

[25] Seohong Park, Jongwook Choi, Jaekyeom Kim, Honglak Lee, and Gunhee Kim. Lipschitz-constrained unsupervised skill discovery. In *International Conference on Learning Representations (ICLR)*, 2021.

[26] Seohong Park, Kimin Lee, Youngwoon Lee, and Pieter Abbeel. Controllability-aware unsupervised skill discovery. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023.

[27] Karl Pertsch, Youngwoon Lee, and Joseph Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on robot learning (CoRL)*, 2021.

[28] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

[29] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations(ICLR)*, 2020.

[30] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

[31] Ruijie Zheng, Yongyuan Liang, Xiyao Wang, Shuang Ma, Hal Daumé III, Huazhe Xu, John Langford, Praveen Palanisamy, Kalyan Shankar Basu, and Furong Huang. Premier-taco is a few-shot policy learner: pretraining multitask representation via temporal action-driven contrastive loss. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[32] Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Russ R Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[33] Michael Laskin, Hao Liu, Xue Bin Peng, Denis Yarats, Aravind Rajeswaran, and Pieter Abbeel. Unsupervised reinforcement learning with contrastive intrinsic control. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[34] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International conference on machine learning (ICML)*, 2020.

[35] Karl Pertsch, Youngwoon Lee, Yue Wu, and Joseph J. Lim. Demonstration-guided reinforcement learning with learned skills. *5th Conference on Robot Learning (CoRL)*, 2021.

[36] Qiyang Li, Jason Zhang, Dibya Ghosh, Amy Zhang, and Sergey Levine. Accelerating exploration with unlabeled prior data. *Advances in Neural Information Processing Systems(NeurIPS)*, 36, 2023.

[37] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on robot learning (CoRL)*, 2020.

[38] 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 *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[39] Yijie Guo, Bingjie Tang, Iretiayo Akinola, Dieter Fox, Abhishek Gupta, and Yashraj Narang. SRSA: Skill retrieval and adaptation for robotic assembly tasks. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[40] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[41] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[42] James Seale Smith, Yen-Chang Hsu, Lingyu Zhang, Ting Hua, Zsolt Kira, Yilin Shen, and Hongxia Jin. Continual diffusion: Continual customization of text-to-image diffusion with c-loRA. *Transactions on Machine Learning Research (TMLR)*, 2024.

[43] Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. Orthogonal subspace learning for language model continual learning. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[44] Yan-Shuo Liang and Wu-Jun Li. Inflora: Interference-free low-rank adaptation for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[45] Yichen Wu, Hongming Piao, Long-Kai Huang, Renzhen Wang, Wanhua Li, Hanspeter Pfister, Deyu Meng, Kede Ma, and Ying Wei. SD-loRA: Scalable decoupled low-rank adaptation for class incremental learning. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[46] Da-Wei Zhou, Hai-Long Sun, Jingyi Ning, Han-Jia Ye, and De-Chuan Zhan. Continual learning with pre-trained models: A survey. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, 2024.

[47] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

[48] Yujia Qin, Cheng Qian, Xu Han, et al. Recyclable tuning for continual pre-training. In *Findings of the Association for Computational Linguistics (ACL)*, 2023.

[49] Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, and Hao Wang. Continual learning of large language models: A comprehensive survey. *arXiv preprint arXiv:2404.16789*, 2024.

[50] W. Aha, Dennis Kibler, and Marc Albert. Instance-based learning algorithms. *Machine Learning*, 1991.

[51] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *International conference on machine learning (ICML)*, 2017.

[52] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[53] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning (ICML)*, 2016.

[54] Kourosh Hakhamaneshi, Ruihan Zhao, Albert Zhan, Pieter Abbeel, and Michael Laskin. Hierarchical few-shot imitation with skill transition models. In *International Conference on Learning Representations (ICLR)*, 2022.

[55] Soroush Nasiriany, Tian Gao, Ajay Mandlekar, and Yuke Zhu. Learning and retrieval from prior data for skill-based imitation learning. In *6th Annual Conference on Robot Learning (CoRL)*, 2022.

[56] Maximilian Du, Suraj Nair, Dorsa Sadigh, and Chelsea Finn. Behavior Retrieval: Few-Shot Imitation Learning by Querying Unlabeled Datasets. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[57] Li-Heng Lin, Yuchen Cui, Amber Xie, Tianyu Hua, and Dorsa Sadigh. Flowretrieval: Flow-guided data retrieval for few-shot imitation learning. In *8th Annual Conference on Robot Learning (CoRL)*, 2024.

[58] Marius Memmel, Jacob Berg, Bingqing Chen, Abhishek Gupta, and Jonathan Francis. STRAP: Robot sub-trajectory retrieval for augmented policy learning. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[59] Kaustubh Sridhar, Souradeep Dutta, Dinesh Jayaraman, and Insup Lee. REGENT: A retrieval-augmented generalist agent that can act in-context in new environments. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[60] Haoqi Yuan, Zhancun Mu, Feiyang Xie, and Zongqing Lu. Pre-training goal-based models for sample-efficient reinforcement learning. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[61] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences of India*, 1936.

[62] Yifeng Zhu, Peter Stone, and Yuke Zhu. Bottom-up skill discovery from unsegmented demonstrations for long-horizon robot manipulation. *IEEE Robotics and Automation Letters*, 2022.

[63] Peter Rousseeuw. Rousseeuw, p.j.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. comput. appl. math. 20, 53-65. *Journal of Computational and Applied Mathematics*, 20:53–65, 11 1987.

[64] Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *6th Conference on robot learning (CoRL)*, 2022.

[65] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[66] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

[67] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning (CoRL)*, 2020.

[68] Byeonghwi Kim, Minhyuk Seo, and Jonghyun Choi. Online continual learning for interactive instruction following agents. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

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

[70] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in neural information processing systems (NeurIPS)*, 2020.

[71] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2023.

[72] Jesse Zhang, Minho Heo, Zuxin Liu, Erdem Biyik, Joseph J Lim, Yao Liu, and Rasool Fakoor. Extract: Efficient policy learning by extracting transferable robot skills from offline data. *arXiv preprint arXiv:2406.17768*, 2024.

[73] Kyowoon Lee, Seongun Kim, and Jaesik Choi. Adaptive and explainable deployment of navigation skills via hierarchical deep reinforcement learning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023.

[74] Hongjoon Ahn, Sungmin Cha, Donggyu Lee, and Taesup Moon. Uncertainty-based continual learning with adaptive regularization. *Advances in neural information processing systems (NeurIPS)*, 32, 2019.

[75] Wonje Choi, Jinwoo Park, Sanghyun Ahn, Daehee Lee, and Honguk Woo. Nesyc: A neuro-symbolic continual learner for complex embodied tasks in open domains. In *The Thirteenth International Conference on Learning Representations (NeurIPS)*, 2025.

[76] Dongsu Lee and Minhae Kwon. Temporal distance-aware transition augmentation for offline model-based reinforcement learning. In *Forty-second International Conference on Machine Learning (ICML)*, 2025.

[77] Sangwoo Shin, Daehee Lee, Minjong Yoo, Woo Kyung Kim, and Honguk Woo. One-shot imitation in a non-stationary environment via multi-modal skill. In *Fortieth International Conference on Machine Learning (ICML)*, 2023.

[78] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

[79] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

Appendix

Contents

A SIL-C Implementation

This section details key notations in A.1 and operational steps in A.2 and throughout the method. We aim to facilitate understanding of concepts described in the main text and supplementary explanations provided in this appendix.

A.1 Notations

This subsection summarizes all notations used throughout the paper.

Table 4: Notation used in the paper

Symbol	Meaning	Symbol	Meaning
MDP fundamentals			
$\mathcal{M}$	$(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mu_0, \gamma)$	$\mathcal{S}$	State space
$\mathcal{A}$	Action space	$\mathcal{P}$	Transition probability
$\mathcal{R}$	Reward function	μ_0	Initial-state distribution
γ	Discount factor $(0 < \gamma < 1)$	s	State
a	Action	$r,\ R$	Step reward; episode return
π	(Stochastic) policy	τ	Task identifier
Hierarchical policy			
π_h	High-level policy	π_l	Low-level skill decoder
z_h	High-level subtask index	z_l	Low-level skill index
θ_h	Parameters of π_h	θ_l	Parameters of π_l
$\mathcal{Z}$	Discrete subtask/skill index space		
Skill Incremental Learning (SIL)			
p	SIL phase index	P	Total # of phases
$\{\mathcal{D}_p\}_{p=1}^{P}$	Datastream (task-agnostic trajectories)	$\mathcal{T}_p$	Evaluation task set at phase p
$\mathcal{Z}_p$	Subtask/skill index set at phase p	$\mathcal{D}_\tau$	Expert demonstrations for task τ
π_h^τ	High-level policy for task τ	π_l^p	Skill decoder at phase p
θ_h^τ	Params of π_h for task τ	θ_l^p	Params of π_l at phase p
Lazy-Learning Interface (bilateral modules & instance based classifier)			
$\mathcal{X}_h$	Subtask space	$\mathcal{X}_l$	skill space
g	Subgoal state	m	Steps ahead to define g
$\Psi_h^s(s; \mathcal{G}_\tau)$	*Task-side* subgoal predictor: returns g	$\mathcal{X}_h^{s,\tau}$	Task memory of τ (subtask prototypes)
$\Psi_l^g(g, z)$	*Skill-side* validator: 1 iff $d_z(g) \leq \delta_z$	$\mathcal{X}_l^{g,p}$	Skill memory at phase p (subgoal-based prototypes)
$\Psi_l^s(s; \mathcal{Z}')$	*Skill-side* classifier (restricted to $\mathcal{Z}'$): returns z_l	$\mathcal{X}_l^{s,p}$	Skill memory at phase p (state-based prototypes)
$\mathcal{I}(s, z_h)$	Interface (Eq. (8))	$\mathcal{Z}'$	Candidate skills s.t. $\Psi_l^g(g, z') = 1$
δ_c	Distance threshold for class c	$d_c(x)$	Mahalanobis distance to class-c prototype
χ_c	Prototype for class c	$\mu_{c,k}, \Sigma_{c,k}$	Mean and (diagonal) covariance of k-th Gaussian
K_c	# of sub-clusters (modes) for χ_c	x	Query instance (e.g., s or g)
Prototype construction (skill & subtask spaces)			
$\mathcal{E}_{\bar{z}}$	Clustered skill group	$\bar{\mathcal{Z}}_p$	Index set of skills discovered at phase p
$\mathcal{H}_{\bar{z},k}^s,\ \mathcal{H}_{\bar{z},k}^g$	Sub-clusters for skill $\bar{z}$ (state/subgoal)	$K_{\bar{z}}$	# of sub-clusters for skill $\bar{z}$
$\chi_{\bar{z}}^s,\ \chi_{\bar{z}}^g$	Prototypes for skill $\bar{z}$ (state/subgoal)	$\mu_{\bar{z},k}, \Sigma_{\bar{z},k}$	Mean, covariance of skill-$\bar{z}$ sub-cluster
$\mathcal{E}_{\bar{g}}$	Clustered subtask group	$\mathcal{G}_\tau$	Subtask label set (from demos)
$\mathcal{H}_{\bar{g},k}^s$	Sub-clusters for subtask of subgoal $\bar{g}$ (state)	$K_{\bar{g}}$	# of sub-clusters for subtask of subgoal $\bar{g}$
$\chi_{\bar{g}}^s$	Subtask prototype (state)	$\mu_{\bar{g},k}, \Sigma_{\bar{g},k}$	Mean, covariance of subtask sub-cluster of subgoal $\bar{g}$

A.2 Algorithms

We provide phase-level training and run-time inference procedures that instantiate the SIL-C framework using the append-only prototype memories and the bilateral lazy-learning interface described in Sec. 4.

Algorithm 1 SIL-C on SIL & Policy Learning with Interface at phase p

```
Require: Phase index $p$, streamed dataset $\mathcal{D}_p$, set of evaluation tasks $\mathcal{T}_p$, demonstrations $\{\mathcal{D}_\tau\}_{\tau\in\mathcal{T}_p}$,
         previous decoder $\pi_l^{(p-1)}$, previous memories $\mathcal{X}_l^{s,(p-1)}$, $\mathcal{X}_l^{g,(p-1)}$, previous index set $\mathcal{Z}_{p-1}$
Ensure: Updated decoder $\pi_l^p$, updated memories $\mathcal{X}_l^{s,p}$, $\mathcal{X}_l^{g,p}$, updated index set $\mathcal{Z}_p$, trained high-
        level policies $\{\pi_h^\tau\}_{\tau\in\mathcal{T}_p}$, task memories $\{\mathcal{X}_h^{s,\tau}\}_{\tau\in\mathcal{T}_p}$
  # SIL AT PHASE-$p$
  # SKILL SPACE UPDATE (EQ. (5))                                                      ▷
  Skill clustering: segment $\mathcal{D}_p$ into $\{\mathcal{E}_{\bar z}\}_{\bar z\in\bar{\mathcal{Z}}_p}$
  Initialize $\bar{\mathcal{X}}_l^{s,p} \leftarrow \emptyset$, $\bar{\mathcal{X}}_l^{g,p} \leftarrow \emptyset$
  for all $\bar z \in \bar{\mathcal{Z}}_p$ do
      Sub-clustering: apply K-means to $\mathcal{E}_{\bar z}$ to obtain $\{\mathcal{H}^s_{\bar z,k}, \mathcal{H}^g_{\bar z,k}\}_{k=1}^{K_{\bar z}}$
      Prototype creation: compute $\{\mu_{\bar z,k}, \Sigma_{\bar z,k}\}_{k=1}^{K_{\bar z}}$ and form $\chi^s_{\bar z}, \chi^g_{\bar z}$
      Add prototypes: $\bar{\mathcal{X}}_l^{s,p} \leftarrow \bar{\mathcal{X}}_l^{s,p} \cup \{\chi^s_{\bar z}\}$,   $\bar{\mathcal{X}}_l^{g,p} \leftarrow \bar{\mathcal{X}}_l^{g,p} \cup \{\chi^g_{\bar z}\}$
  Memory append: $\mathcal{X}_l^{s,p} \leftarrow \mathcal{X}_l^{s,(p-1)} \cup \bar{\mathcal{X}}_l^{s,p}$,   $\mathcal{X}_l^{g,p} \leftarrow \mathcal{X}_l^{g,(p-1)} \cup \bar{\mathcal{X}}_l^{g,p}$
  # LOW-LEVEL SKILL DECODER UPDATE                                                    ▷
  Train $\pi_l$ with $A_l$ on $\mathcal{D}_p$ to obtain $\pi_l^p$
  Index growth: $\mathcal{Z}_p \leftarrow \mathcal{Z}_{p-1} \cup \bar{\mathcal{Z}}_p$

  # DOWNSTREAM POLICY TRAINING AT PHASE-$p$
  for all $\tau \in \mathcal{T}_p$ do
      # SUBTASK SPACE UPDATE (EQ. (6))                                                ▷
      Initialize $\mathcal{X}_h^{s,\tau} \leftarrow \emptyset$
      Subtask clustering: segment $\mathcal{D}_\tau$ into $\{\mathcal{E}_{\bar g}\}_{\bar g\in\mathcal{G}_\tau}$
      for all $\bar g \in \mathcal{G}_\tau$ do
          Sub-clustering: apply K-means to $\mathcal{E}_{\bar g}$ to obtain $\{\mathcal{H}^s_{\bar g,k}\}_{k=1}^{K_{\bar g}}$
          Prototype creation: compute $\{\mu_{\bar g,k}, \Sigma_{\bar g,k}\}_{k=1}^{K_{\bar g}}$ and form $\chi^s_{\bar g}$
          Add prototypes: $\mathcal{X}_h^{s,\tau} \leftarrow \mathcal{X}_h^{s,\tau} \cup \{\chi^s_{\bar g}\}$
      # LABEL ASSIGNMENT WITH UPDATED DECODER (EQ. (7))                               ▷
      for all transition $(s, a^*) \in \mathcal{D}_\tau$ do
          $z_h^* \leftarrow \arg\min_{z_l\in\mathcal{Z}_p} \|\hat a - a^*\|_2^2, \quad \hat a \sim \pi_l^p(\cdot \mid s, z_l)$
      # HIGH-LEVEL POLICY OPTIMIZATION                                                ▷
      Update $\pi_h^\tau$ by minimizing $\mathbb{E}_{(s,z_h^*)}[-\log \pi_h^\tau(z_h^* \mid s)]$
  return $(\pi_l^p,\ \mathcal{X}_l^{s,p},\ \mathcal{X}_l^{g,p},\ \mathcal{Z}_p,\ \{\pi_h^\tau\}_{\tau\in\mathcal{T}_p},\ \{\mathcal{X}_h^{s,\tau}\}_{\tau\in\mathcal{T}_p})$
```

Phase training (Alg. 1). *(A) Skill space update.* At the beginning of phase p, the streamed dataset $\mathcal{D}_p$ is segmented into skill groups $\{\mathcal{E}_{\bar z}\}_{\bar z\in\bar{\mathcal{Z}}_p}$ ((5)). For each group, we perform K-means in the state and subgoal spaces to obtain sub-clusters $\{\mathcal{H}^s_{\bar z,k}, \mathcal{H}^g_{\bar z,k}\}$ and form Gaussian prototypes $\chi^s_{\bar z}, \chi^g_{\bar z}$ with diagonal covariance. Newly created prototypes are *appended* to the skill memories $\mathcal{X}_l^{s,p}, \mathcal{X}_l^{g,p}$ without altering past entries, and the skill decoder is updated to π_l^p; the index set grows append-only as $\mathcal{Z}_p \leftarrow \mathcal{Z}_{p-1} \cup \bar{\mathcal{Z}}_p$. *(B) Policy learning on the interface.* For each task τ, we build the subtask memory by clustering $\mathcal{D}_\tau$ into $\{\mathcal{E}_{\bar g}\}_{\bar g\in\mathcal{G}_\tau}$ and constructing state-side prototypes $\{\chi^s_{\bar g}\}$ ((6)). Labels for behavior cloning are assigned by the energy-based prior that selects z_h^* minimizing the decoder mismatch ((7)), and π_h^τ is trained with cross-entropy over (s, z_h^*).

Algorithm 2 SIL-C Evaluation with Interface

```
Decoder $\pi_l^p$, policy $\pi_h^\tau$, interface modules $\Psi_l^{s,p}$, $\Psi_l^{g,p}$, $\Psi_h^{s,\tau}$,
task memory $\mathcal{G}_\tau$, skill index set $\mathcal{Z}_p$, initial state $s_0$
Initialize $R \leftarrow 0$,  $s \leftarrow s_0$
while episode not terminated do
    # HIGH-LEVEL POLICY
    $z_h \sim \pi_h^\tau(z_h \mid s)$                                  ▷ sample subtask
    # INTERFACE INFERENCE (SUBTASK SPACE)
    $g \leftarrow \Psi_h^{s,\tau}(s; \mathcal{G}_\tau)$                             ▷ predict subgoal
    # INTERFACE INFERENCE (SKILL SPACE) (EQ. (8))
    if $\Psi_l^{g,p}(g, z_h) = 1$ then                                ▷ skill validation
        $z_l \leftarrow z_h$
    else                                                              ▷ skill hooking
        $\mathcal{Z}' \leftarrow \{\, z' \in \mathcal{Z}_p \mid \Psi_l^{g,p}(g, z') = 1 \,\} \cup \{z_h\}$
        $z_l \leftarrow \Psi_l^{s,p}(s; \mathcal{Z}')$
    # LOW-LEVEL SKILL DECODER
    $a \sim \pi_l^p(a \mid s, z_l)$                                    ▷ sample action
    Execute $a$; observe reward $r$ and next state $s'$;  $R \leftarrow R + r$;  $s \leftarrow s'$
return $R$
```

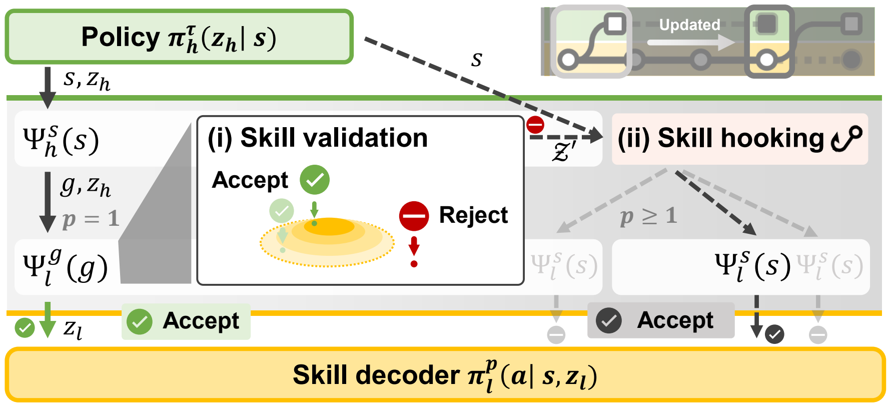


Figure 7: Visualization of agent inference procedure with SIL-C

Agent Inference (Algoritm 2, Figure 7). Given the current state s, the high-level policy samples a subtask $z_h \sim \pi_h^\tau(z \mid s)$, and the task-side module predicts a subgoal $g = \Psi_h^s(s; \mathcal{G}_\tau)$. The interface first performs *skill validation* with the skill-side validator Ψ_l^g: if $\Psi_l^g(g, z_h) = 1$, the subtask is accepted ($z_l = z_h$); otherwise, *skill hooking* restricts the candidate set to $\mathcal{Z}' = \{\, z' \in \mathcal{Z}_p \mid \Psi_l^g(g, z') = 1 \,\} \cup \{z_h\}$ and chooses $z_l = \Psi_l^s(s; \mathcal{Z}')$ (*restricted classification*). Equivalently, validation can be written as $d_z(g) \leq \delta_z$ using the Mahalanobis distance in Eq. (4). This two-stage instance-based procedure operationalizes Eq. (8) and realizes compatibility at inference without policy re-training.

A.3 Pseudocode

```python
## Initialization
from collections import defaultdict
from scipy.stats import chi2

Interface = {
    "skills":   defaultdict(lambda: {"subgoal": [], "state": []}),
    "subtasks": defaultdict(lambda: defaultdict(list)),
} # Gaussian component = (mean, precision)
pi_l, pi_h = None, {}

def label_dist(x, comps): # Eq.(4)
    def maha(x, comp):
        mu, Sigma_inv = comp
        d = x - mu
        return d.T @ Sigma_inv @ d
    return min(maha(x, c) for c in comps)

def chi2_thr(comps, conf=0.99):
    dim = comps[0][0].shape[0]
    return chi2.ppf(conf, dim)

## Policy compatible skill incremental learning
for p in range(1, P + 1):
    # Skill space update Eq.(5)
    chi_gz, chi_sz = create_skill_prototypes(cluster_skills(D_p))
    for z in chi_gz:
        Interface["skills"][z]["subgoal"].extend(chi_gz[z])
        Interface["skills"][z]["state"].extend(chi_sz[z])
    # Skill incremental learning
    pi_l = train_low_level_decoder(D_p, Interface["skills"], pi_l)

    # Downstream policy learning
    for tau in T_p:
        # Subtask space update Eq.(6)
        chi_g = create_subtask_prototypes(cluster_subtasks(D_tau))
        for g in chi_g:
            Interface["subtasks"][tau][g].extend(chi_g[g])

        # Policy learning on the interface Eq.(7)
        labels = assign_subtask_labels(D_tau, Interface["skills"], pi_l)
        pi_h[tau] = train_high_level_policy(D_tau, labels)

## Policy inference via the interface
def infer_action(s, tau, conf=0.99):
    # High-level policy: subtask proposal
    z_h = pi_h[tau](s)
    g   = predict_subgoal(s, Interface["subtasks"][tau])
    delta = chi2_thr(Interface["skills"][z_h]["subgoal"], conf)

    # Interface: skill validation and skill hooking Eq.(8)
    if label_dist(g, Interface["skills"][z_h]["subgoal"]) <= delta:
        z_l = z_h
    else:
        cand = [z for z in Interface["skills"]
            if label_dist(g, Interface["skills"][z]["subgoal"]) <= delta] + [z_h]
        z_l  = min(cand, key=lambda z:
            label_dist(s, Interface["skills"][z]["state"]))

    # Low-level decoder: action execution
    return pi_l(s, z_l)
```

Figure 8: Python style pseudo code of SIL-C

We provide Python-style pseudo code in Figure 8 to illustrate the SIL-C implementation. The code shows how the append-only prototype memories are constructed and updated during each phase, and how the bilateral interface performs skill validation and hooking at inference time using Mahalanobis distance matching.

A.4 Hyperparameters

We apply a consistent set of hyperparameters to update each space in the SIL-C interface for both the Kitchen and Meta-World environments. The configuration, used in Table 1, is summarized in Table 5.

Table 5: Default hyperparameter configuration for SIL-C

Skill-Side (per phase)		**Task-Side (per task)**	
Sub-clusters per skill ($K_{\bar{z}}$)	4	Sub-clusters per sub-task ($K_{\bar{g}}$)	4
Goal offset (m)	20	Goal offset (m)	20

B Experimental Settings

B.1 Environments

Franka Kitchen [65] is a long-horizon manipulation environment where a robot must complete multi-stage tasks by interacting with various kitchen objects such as a `microwave, kettle, burners, light switches, and cabinet doors`. We use the `kitchen-mixed-v0` dataset for experiment. Each demonstration consists solely of state-action transitions, without any reward signals or task labels. The state space is 60-dimensional, encompassing both the robot arm state and the states of the manipulated objects. The action space is 9-dimensional, corresponding to the 9 degrees of freedom (DoF) of the Franka robotic arm. Each task requires the robot to complete a sequence of 4 subtasks(stage) selected from a fixed set of 7 predefined subtasks: `open microwave, move kettle, turn on top burner, turn on bottom burner, light switch, open hinge cabinet, open slide cabinet`. A reward of 1.00 is given for each subtask successfully completed in the correct order, yielding a maximum of 4.00 when all subtasks are completed. For evaluation, we normalize the reward such that the maximum score is 100.

Multi-stage Meta-World extends the original Meta-World benchmark [67], which includes 50 diverse single-step robot tasks, by composing multiple tasks into sequential stages within a single episode. Each multi-stage task requires an agent to complete 4 subtasks in a fixed order, mimicking realistic long-horizon objectives `puck, drawer, button, door, box, handle, lever, stick` [77]. We use the `Easy` dataset introduced by [11], which consists of composed tasks involving 4 objects selected from a subset of: `puck, drawer, button, door`, with varied subtask sequences. Each demonstration consists solely of state-action transitions, without any reward signals or task labels. The state space is 140-dimensional, capturing both the robot arm state and the states of the relevant objects. The action space is 4-dimensional, corresponding to the 4 degrees of freedom of the robotic arm. Each task consists of 4 sequential subtasks selected from the predefined set: `slide puck`, `close drawer`, `push button`, `open door`, `close box`, `press handle`, `pull lever`, and `insert stick into red box`. As in the Franka Kitchen environment, a reward of 1.00 is given for each subtask completed in the correct order, with a maximum cumulative reward of 4.00 per task. For evaluation, we normalize the reward so that the maximum possible score is 100.

B.2 SIL Scenario

In each phase p, a new skill dataset $\mathcal{D}_p$ is provided to train the skill decoder. Subsequently, based on the trained low-level skill decoder, 24 task-specific high-level policies are individually trained.

Table 6: Kitchen: *Emergent SIL*

Kitchen : *Emergent SIL*				
Phase	**Skill Dataset**			
1	microwave	bottom burner	light switch	slide cabinet
	bottom burner	top burner	light switch	slide cabinet
	microwave	bottom burner	top burner	light switch
	microwave	bottom burner	slide cabinet	hinge cabinet
	microwave	bottom burner	top burner	slide cabinet
	bottom burner	top burner	slide cabinet	hinge cabinet
	kettle	bottom burner	top burner	slide cabinet
	microwave	bottom burner	top burner	hinge cabinet
2	microwave	kettle	bottom burner	hinge cabinet
	microwave	kettle	top burner	hinge cabinet
	kettle	bottom burner	top burner	hinge cabinet
	microwave	kettle	top burner	light switch
	microwave	top burner	light switch	hinge cabinet
	microwave	kettle	light switch	hinge cabinet
3	microwave	light switch	slide cabinet	hinge cabinet
	microwave	kettle	slide cabinet	hinge cabinet
	microwave	kettle	bottom burner	slide cabinet
	microwave	kettle	light switch	slide cabinet
4	kettle	top burner	light switch	slide cabinet
	kettle	bottom burner	slide cabinet	hinge cabinet
	kettle	bottom burner	light switch	hinge cabinet
	kettle	bottom burner	top burner	light switch
	kettle	light switch	slide cabinet	hinge cabinet
	kettle	bottom burner	light switch	slide cabinet

Table 7: Meta-World: *Emergent SIL*

Meta-World: *Emergent SIL*				
Phase	**Skill Datasets**			
1	puck	drawer	button	door
	puck	drawer	door	button
	puck	button	drawer	door
	puck	button	door	drawer
	puck	door	drawer	button
	puck	door	button	drawer
2	drawer	puck	button	door
	drawer	puck	door	button
	drawer	button	puck	door
	drawer	button	door	puck
	drawer	door	puck	button
	drawer	door	button	puck
3	button	puck	drawer	door
	button	puck	door	drawer
	button	drawer	puck	door
	button	drawer	door	puck
	button	door	puck	drawer
	button	door	drawer	puck
4	door	puck	drawer	button
	door	puck	button	drawer
	door	drawer	puck	button
	door	drawer	button	puck
	door	button	puck	drawer
	door	button	drawer	puck

Table 8: Kitchen: *Explicit SIL*

Kitchen: *Explicit SIL*	
Phase	**Skill Datasets**
1	microwave
2	kettle / bottom burner
3	top burner / light switch
4	slide cabinet / hinge cabinet

Table 9: Meta-World: *Explicit SIL*

Meta-World: *Explicit SIL*	
Phase	**Skill Datasets**
1	puck
2	drawer
3	button
4	door

B.2.1 Datastream Types : Emergent and Explicit SIL

Emergent SIL. In the *Emergent SIL* scenario, we divide each dataset into four datastreams by grouping full task-agnostic demonstrations, resulting in datastreams that contain a mixture of tasks without explicit task identifiers. To facilitate scenario construction and explanation, we partition the dataset using task information. To maintain the task-agnostic nature of skill learning, this information is not exposed during skill-incremental learning, and the learning process remains fully task-agnostic. In the case of **Franka Kitchen**, we collect task demonstrations based on the predefined subtask goal information provided by the environment. Failed or truncated trajectories are discarded, and we identify 24 distinct tasks. The resulting datastream composition is summarized in Table 6. For **Meta-World**, we follow a similar approach, segmenting the dataset into 24 tasks, corresponding to all 4! possible subtask sequence permutations, by verifying subtask goal completion using the environment's built-in success conditions. The datastream configuration is shown in Table 7.

Explicit SIL. In the *Explicit SIL* scenario, we segment each dataset into shorter demonstrations based on predefined skill boundaries specified by the environment. These skill segments are then grouped into four datastreams. To facilitate scenario construction and explanation, we use skill labels provided by the environment to organize the data. However, this information is not exposed to the agent during training, and the learning process remains fully skill-agnostic. For **Franka Kitchen**, we segment trajectories by identifying transitions that correspond to each predefined subtask. These are clustered by subtask type to form skill-specific datastreams, as shown in Table 8. For **Meta-World**, we apply a similar segmentation strategy, isolating the trajectory from the beginning of each subtask to the point where the agent returns to a neutral position. These segments are then clustered according to subtask identity, resulting in the skill incremental setup summarized in Table 9.

B.2.2 Evaluation Groups for SIL Scenarios

At each phase p, a new skill dataset D_p is used to train the skill decoder. Using this updated decoder, we then train 24 task-specific high-level policies, one for each evaluation task. Across both environments, we evaluate all 24 tasks listed in Table 6 and Table 7 by training a dedicated policy for each task. This setup enables us to evaluate bidirectional compatibility. Specifically, we measure *Backward Skill Compatibility* (BwSC) by checking whether policies trained in earlier phases remain functional with updated skills. In parallel, we assess *Forward Skill Compatibility* (FwSC) by determining whether new policies can effectively leverage skills learned in previous phases.

Formally, the evaluation groups for BwSC_τ and FwSC_τ of task τ are defined as:

$$\text{BwSC}_\tau = [(\pi_h^{\tau,(1)}, \pi_l^{(1)}), (\pi_h^{\tau,(1)}, \pi_l^{(2)}), \ldots, (\pi_h^{\tau,(1)}, \pi_l^{(p)})] \tag{9}$$

$$\text{FwSC}_\tau = [(\pi_h^{\tau,(1)}, \pi_l^{(1)}), (\pi_h^{\tau,(2)}, \pi_l^{(2)}), \ldots, (\pi_h^{\tau,(p)}, \pi_l^{(p)})] \tag{10}$$

Here, $\pi_h^{\tau,(1)}$ denotes the initial high-level policy trained with the skill decoder from the first phase $p = 1$. For any phase $p \geq 1$, $\pi_h^{\tau,(p)}$ refers to the updated high-level policy synchronized with the corresponding skill decoder $\pi_l^{(p)}$ at phase p.

B.3 Baseline Details

We categorize our baselines into four distinct types based on their methodology and compatibility with the Skill Incremental Learning (SIL) scenario.

Type I: Skill-based approaches with continual learning. Type I combines skill-based pre-training with continual learning strategies using a simple skill-space appending scheme. We adopt two hierarchical skill pre-training methods:

- **BUDS** [62, 10]: BUDS [62] segments trajectories into fixed-length intervals and merges them bottom-up based on trajectory similarity, and was later adopted as the foundation in LOTUS [10] for continual imitation learning. It discovers skills by clustering these segments. In our implementation, we base our hyperparameter choices on those reported in BUDS. To support goal-conditioned skill decoder learning, we annotate each trajectory with subgoal states prior to skill discovery. For each discovered skill, we select its goal representation as the subgoal state closest to the average of the subgoals observed across its transitions. Each phase can generate up to 10 skills, but the actual number is determined automatically using the silhouette score. In the Kitchen environment, this resulted in an average of approximately 8 skills per phase.
- **PTGM** [60]: PTGM leverages subgoal state information during skill decoding by discretizing the subgoal state space and treating each subgoal bin as a distinct skill. This approach supports pre-training in open domains and demonstrates strong performance on downstream tasks, particularly in complex environments, often outperforming skill pre-training methods based on continuous skill representations. In our experiments, we adopt the hyperparameters from the PTGM paper for the Kitchen domain. Each phase produces a total of 20 skill clusters.

In both methods, we define the subgoal state for each transition as the state reached after $m = 20$ steps. To support the SIL scenario, we expand the skill set at each phase by appending newly discovered skills to those from previous phases. This expansion is strictly append-only and does not modify previously learned skills.

These pre-trained skills are integrated with continual learning or adaptation strategies as follows:

- **Fine-Tuning (FT)**: The simplest approach. The skill decoder is incrementally trained on each new skill without access to prior data, no memory or replay is used.

- **Experience Replay (ER)** [69]: In our implementation, a replay buffer stores 10% of the data from each previous phase. During training in the next phase, these stored samples are interleaved with new data at a 1:1 sampling ratio to mitigate forgetting.
- **Adapter Append (AA)** [38]: In the first phase, we pre-train the entire model. Starting from the second phase, we freeze the pre-trained model and train a separate LoRA adapter [78] for each new phase, using rank 4 for the *Emergent* scenario and rank 16 for the *Explicit* scenario. During evaluation, we use the adapter corresponding to the phase in which the skill was introduced. This strategy, introduced in TAIL [38], preserves skill-policy compatibility by preventing forgetting in both the base model and its adapted parameters. However, it limits the ability to transfer or incorporate newly acquired skills across different phases.

Type II: Semantic representation-based skill incremental learning approaches. These methods rely on predefined semantic subgoals as skill labels. They incorporate prototype-based skill retrieval and model expansion with temporal replay.

- **Skill-prototypes + (AA)** [11]: Performs skill retrieval using a prototype memory, where each prototype is linked with adapter parameters and indexed by persistent semantic skill labels. We set the number of skill prototype bases to 50. Following the original setup, tasks are learned using semantic subgoal labels provided in a fixed sequence.
- **Instructions + (ER)** [12]: Defines and learns skills incrementally from semantic representations. We adopt its trajectory-based temporal replay to support skill incremental learning.

To ensure a fair comparison, these methods are restricted to using only the semantically labeled skills available for policy training at the corresponding SIL phase p. Trajectories without semantic skill labels are merged with the skill from the previous transition.

Type III: SIL-C. Our method builds on the Type I structure and its configuration. In our SIL setup, we assign four sub-clusters to each skill prototype. For subtask prototype construction, we follow the PTGM algorithm, setting 20 prototypes per task and assigning four sub-clusters to each prototype as well.

High-level policy and skill decoder implementation. For all baselines, we implement the high-level policy as a four-layer MLP that acts as a classifier for sub-task prediction. To enable behavior cloning in SIL-C, we generate supervision signals using Eq. (7) and train the classifier with cross-entropy loss.

The skill decoder, used across all baselines, is a goal-conditioned policy consisting of two components. The first maps each skill to a hashed embedding. Following the skill decoder architecture proposed in [60], we compute each skill embedding as the subgoal state closest to the centroid of the subgoal states comprising that skill. We then feed this subgoal state, along with the current state, into a conditional denoising diffusion model with four conditioned denoising blocks. The model denoises from Gaussian noise to reconstruct the corresponding action.

For Type II methods, where skills are defined semantically (e.g., using natural language descriptions), we embed the descriptions using the `text-embedding-3-large` model from OpenAI and use these vectors directly as skill representations.

Table 10: Default policy configuration

Hyperparameter	Value
Model	MLP
Hidden size	512
Hidden layers	4
Dropout	0.1
State input dim	60
Optimizer	Adam
Learning rate	1×10^{-4}
β_1	0.9

Table 11: Default skill decoder configuration

Hyperparameter	Value
Diffusion Model	DDPM [70]
Denoising step	64
Schedule	Linear
Linear start	1e-4
Linear end	2e-2
Block	MLP
Hidden dimension	512
Layers	4
Dropout	0.0
Clip denoised	True
Optimizer	Adam
Learning rate	2×10^{-5}
β_1	0.9

B.4 Metrics

For each scenario, we evaluate the *FwSC* and *BwSC* groups using three metrics: Forward Transfer (FWT), Backward Transfer (BWT), and Area Under the Curve (AUC). *Overall performance* is reported by computing each metric over the union of both groups. Let $r^{\tau,p}$ denote the normalized reward for task τ at phase p, evaluated using the corresponding high-level policy and skill decoder. For example, $r^{\tau,p} = (\pi_h^{\tau,(1)}, \pi_l^p)$ for the *BwSC* group, and $r^{\tau,p} = (\pi_h^{\tau,p}, \pi_l^p)$ for the *FwSC* group. This score reflects task performance under the given high-level and low-level policy configuration. We assume a fixed evaluation task set $\mathcal{T}_1$, which is used across all phases, i.e., $\mathcal{T}_p = \mathcal{T}_1$ for all $p \in \{1, \dots, P\}$.

Forward Transfer (FWT) measures the performance of newly trained policies after skill updates, quantifying how updated skills facilitate learning new tasks.

$$\text{FWT}^{\tau,p} = r^{\tau,p} \tag{11}$$

We report both the *Initial FWT*, computed at $p = 1$, and the *Final FWT*, computed at the final phase $p = P$.

Backward Transfer (BWT) quantifies the influence of skill incremental learning on previously acquired task performance. It is defined as the average change in performance across phases, relative to phase 1:

$$\text{BWT}^{\tau} = \frac{1}{P-1} \sum_{p=2}^{P} \left(r^{\tau,p} - r^{\tau,(1)} \right) \tag{12}$$

Area Under the Curve (AUC) captures the overall performance trend across all skill learning phases, computed as the mean normalized reward:

$$\text{AUC}^{\tau} = \frac{1}{P} \sum_{p=1}^{P} r^{\tau,p} \tag{13}$$

We report the final values for each metric by averaging over all evaluated tasks $\tau \in \mathcal{T}$.

B.5 Training Details

Compute Resources We conducted our experiments in the following computing environments:

- AMD Ryzen 9 7950X3D 16-Core Processor with a single RTX 4090 GPU. OS: Ubuntu 22.04, CUDA Version: 12.4, Driver Version: 550.144.03

- AMD Ryzen Threadripper PRO 5975WX 32-Core CPU with 2× RTX 4090 GPUs. OS: Ubuntu 22.04, CUDA Version: 12.4, Driver Version: 550.144.03
- AMD Ryzen Threadripper PRO 5975WX 32-Core CPU with 2× RTX 4090 GPUs. OS: Ubuntu 22.04, CUDA Version: 12.2, Driver Version: 535.230.02

Training Framework Details We implemented our framework using JAX [79] to efficiently accelerate training and handle the continual phase's alternating training and evaluation process. All software dependencies are included with the released code for full reproducibility. The versions of core libraries are:

- `jax`: 0.4.34
- `jaxlib`: 0.4.34
- `flax`: 0.10.2
- `optax`: 0.1.9

Experiment Compute Estimates For experiments on Kitchen and Meta-World, we ran 4 seeds in parallel (4 processes on a single GPU). Each run took approximately 4 hours and 30 minutes, broken down as follows:

- **Training**: 1 hour for training 4 skill update phases, covering a total of 96 policy training runs (24 tasks per phase).
- **Evaluation**: 3 hours and 30 minutes for 168 task evaluations (24 tasks $\times$ [3 *BwSC* + 4 *FwSC*]).

Total Compute Usage

- **Main experiments only:** Approximately 225 GPU hours (RTX 4090), including core experiments and additional SIL scenario evaluations.
- **Experiments including Appendix:** Approximately 500 GPU hours (RTX 4090), covering extended experiments and additional SIL evaluations.
- **All experiments, including preliminary and discarded runs:** An estimated 600 GPU hours (RTX 4090), accounting for exploratory and failed runs not included in the final results.

B.6 Evaluation Details

To report performance metrics, we used *four* random seeds and report the mean and standard deviation across them. For each scenario, the normalized reward of a given task was computed by running the evaluation three times and averaging the results. The averaged value was recorded as the reward for that task.

Table 12: Performance evaluation of skill-policy compatibility in two SIL scenarios of the Franka-Kitchen environment with four seeds. Each row corresponds to a baseline, categorized by the skill interface configuration, hierarchical agent structure, and the SIL algorithm used for skill decoder updates. The symbol * denotes methods that require pre-defined semantic skill labels for both policy and decoder training. The left side reports *BwSC* results, the right side reports *FwSC* results, and the center shows the overall performance considering both.

Kitchen : *Emergent SIL*										
Baselines			*Initial*	*BwSC* (*Initial*)		Overall performance		*FwSC* (*Synced*)		*Final*
Type	Skill Interface	SIL Algo.	FWT (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	FWT (%)
I	Skill Segments [62, 10] (BUDS)	FT	$23.5_{\pm 4.9}$	$-16.4_{\pm 5.8}$	$11.2_{\pm 1.2}$	$-6.1_{\pm 4.9}$	$18.3_{\pm 1.2}$	$4.3_{\pm 4.3}$	$26.7_{\pm 2.3}$	$18.7_{\pm 1.2}$
		ER	$23.5_{\pm 3.8}$	$-10.8_{\pm 8.1}$	$15.5_{\pm 2.3}$	$3.6_{\pm 7.0}$	$26.6_{\pm 2.8}$	$17.9_{\pm 6.5}$	$37.0_{\pm 3.0}$	$44.5_{\pm 7.0}$
		AA	$21.9_{\pm 4.1}$	$1.3_{\pm 4.4}$	$22.9_{\pm 4.0}$	$15.7_{\pm 3.5}$	$35.3_{\pm 2.9}$	$30.1_{\pm 3.9}$	$44.4_{\pm 2.6}$	$57.1_{\pm 7.5}$
	Subgoal Bins [60] (PTGM)	FT	$45.0_{\pm 2.0}$	$-36.9_{\pm 1.5}$	$17.3_{\pm 0.9}$	$-24.3_{\pm 1.6}$	$24.1_{\pm 1.0}$	$-11.7_{\pm 2.1}$	$36.2_{\pm 1.6}$	$24.7_{\pm 2.0}$
		ER (10%)	$46.8_{\pm 2.5}$	$-19.7_{\pm 4.4}$	$32.1_{\pm 2.9}$	$-5.0_{\pm 2.8}$	$42.5_{\pm 1.4}$	$9.7_{\pm 3.3}$	$54.0_{\pm 1.8}$	$57.7_{\pm 3.6}$
		ER (50%)	$48.2_{\pm 1.9}$	$-9.2_{\pm 2.2}$	$41.3_{\pm 2.0}$	$5.8_{\pm 1.8}$	$53.1_{\pm 1.8}$	$20.8_{\pm 1.9}$	$63.8_{\pm 1.7}$	$74.7_{\pm 2.2}$
		MT (100%)	$48.4_{\pm 1.4}$	$-3.1_{\pm 1.8}$	$46.1_{\pm 0.9}$	$11.6_{\pm 0.9}$	$58.4_{\pm 1.2}$	$26.2_{\pm 0.9}$	$68.1_{\pm 1.7}$	$81.3_{\pm 4.5}$
		AA	$45.0_{\pm 3.1}$	$2.6_{\pm 0.9}$	$46.9_{\pm 3.7}$	$15.4_{\pm 1.7}$	$58.2_{\pm 1.7}$	$28.1_{\pm 4.1}$	$66.1_{\pm 0.6}$	$83.6_{\pm 3.1}$
II	Skill-prototypes [11]*	AA	$55.1_{\pm 1.8}$	$0.5_{\pm 3.1}$	$55.5_{\pm 2.7}$	$0.5_{\pm 2.8}$	$55.7_{\pm 2.7}$	$0.9_{\pm 2.2}$	$55.8_{\pm 2.4}$	$56.9_{\pm 4.9}$
	Instructions [12]*	ER	$54.0_{\pm 2.4}$	$18.5_{\pm 4.0}$	$67.8_{\pm 1.1}$	$19.1_{\pm 2.7}$	$70.3_{\pm 1.0}$	$19.7_{\pm 2.0}$	$68.7_{\pm 1.9}$	$77.4_{\pm 2.8}$
III	SIL-C	MT (100%)	$52.7_{\pm 2.8}$	$15.2_{\pm 2.8}$	$64.1_{\pm 2.6}$	$19.7_{\pm 2.1}$	$69.6_{\pm 2.1}$	$24.3_{\pm 1.7}$	$70.9_{\pm 1.7}$	$82.7_{\pm 0.7}$
		AA	$52.9_{\pm 2.9}$	$18.6_{\pm 1.2}$	$66.8_{\pm 2.6}$	$22.0_{\pm 2.4}$	$71.8_{\pm 1.3}$	$25.4_{\pm 4.0}$	$71.9_{\pm 0.9}$	$87.2_{\pm 3.2}$
-	Sub-goal Bins [60]	Joint	-	-	-	-	-	-	-	$86.9_{\pm 2.2}$

C Additional Experiments

C.1 Motivating Examples

Table 12 extends our main results with additional baselines to demonstrate a critical challenge in skill incremental learning: even multi-task learning, often considered an oracle baseline in continual learning, fails to achieve backward skill compatibility (*BwSC*).

Multi-task learning fails to enable compositional skill reuse. We evaluate PTGM with varying replay ratios (10%, 50%, 100%) to understand the limits of experience replay in SIL. Even with 100% replay, where the model has access to all previous data, PTGM achieves only -3.1% BWT in the BwSC setting. This negative transfer indicates that despite having full access to historical data, the method cannot leverage newly acquired skills to improve existing policies without complete retraining. The pattern persists across all replay configurations: ER (10%) shows -19.7% BWT, ER (50%) shows -9.2% BWT, and even multi-task learning (MT 100%) exhibits negative transfer. These results reveal a fundamental architectural limitation: naive combinations of continual learning approaches with skill-based pre-training fail to support dynamic skill composition.

SIL-C enables true compositional learning without retraining. In contrast, SIL-C achieves substantial positive backward transfer across both continual learning strategies: 15.2% BWT with MT (100%) and 18.6% BWT with AA. This improvement is not merely incremental but represents a qualitative difference in capability. While baseline methods require complete policy retraining to utilize new skills (as evidenced by their improved *FwSC* but poor *BwSC* performance), SIL-C's lazy learning interface enables existing policies to dynamically compose newly acquired skills at inference time. This compositional property is particularly evident in the overall performance metrics, where SIL-C maintains competitive or superior AUC scores on *BwSC* (64.1% with MT, 66.8% with AA) while simultaneously achieving the highest final performance (82.7% and 87.2% FWT respectively).

The significance of these results extends beyond numerical improvements: they demonstrate that SIL-C fundamentally changes how skills and policies interact in hierarchical reinforcement learning, enabling true lifelong learning where each new skill enhances the agent's entire behavioral repertoire without the computational burden of retraining.

C.2 Skill-Policy Compatibility : Overview and Ordering Effects

Figure 9 presents phase-wise normalized rewards across four SIL datastreams, each defined by a different permutation of skill datasets from Table 1 (**Seq. 1**). The corresponding evaluation metrics for each stream are summarized in Table 13. These experiments assess both backward skill compatibility

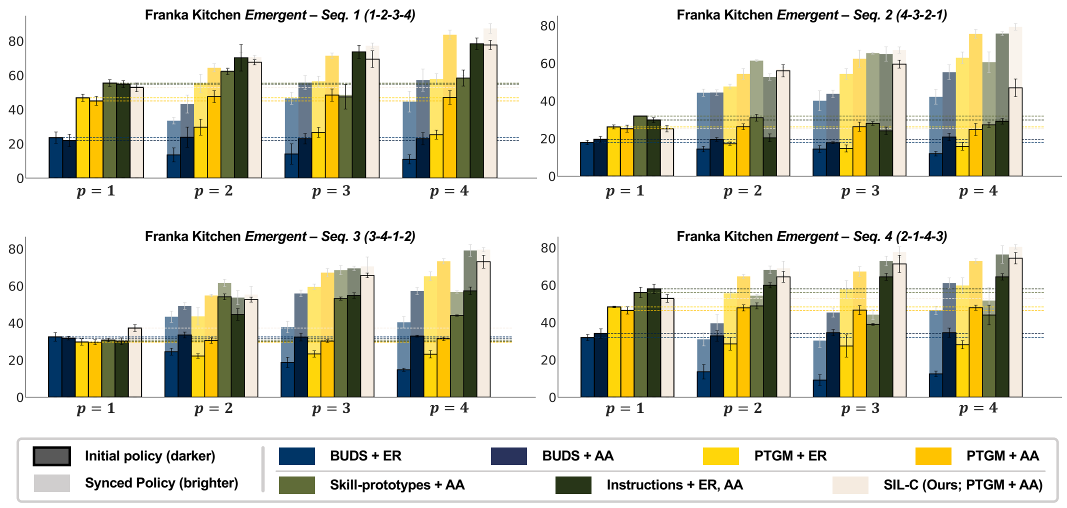


Figure 9: Phase-wise normalized rewards across four Kitchen *Emergent* Skill Incremental datastreams, each corresponding to a different permutation of skill execution order. Each bar group shows performance for different methods under both initial and retrained (Synced) policies.

(*BwSC*), which captures the evolution of the initial policy over phases, and forward skill compatibility (*FwSC*), measured after the skill policies have been resynchronized and retrained. In the *BwSC* setting, SIL-C consistently achieves the highest normalized rewards or matches the performance of methods that use pre-defined semantic skills. After retraining(Synced Policy) in the *FwSC* setting, SIL-C continues to show the highest performance across all phases.

Each sequence corresponds to a permutation of phase orders from Table 6. In **Seq. 2**, the dataset of first phase does not include demonstrations for `open microwave`, so even methods that rely on pre-defined semantic skills fail to execute this task in early phases. Once policies are retrained, SIL-C, SIL with semantic skills, and PTGM+AA reach similar levels of performance. A similar issue appears in **Seq. 3**, where demonstrations for `turn on top burner` are absent in the initial phase. Only SIL-C maintains high performance throughout, showing strong skill-policy compatibility despite the delayed exposure to this skill. **Seq. 4** follows the same pattern for `open slide cabinet`, where again only SIL-C maintains high rewards both before and after retraining.

Additionally, methods that combine semantic retrieval with skill prototypes exhibit decreasing performance over successive phases, particularly in the *Emergent* skill incremental setting. As the diversity of skills increases, these methods are more susceptible to incorrect skill retrieval, which in some cases results in performance degradation relative to earlier phases. Overall, SIL-C is the only method that consistently maintains skill-policy compatibility without requiring prior knowledge of future skills. This property is essential for scalable, SIL.

Table 13: Additional experiments for testing robustness to incremental ordering. Experiments were conducted on the Franka Kitchen environment using various combinations of phase orderings. Each row represents a baseline method, organized according to the skill interface setting, hierarchical agent architecture, and the specific SIL algorithm applied for skill decoder updates. The * symbol indicates methods requiring pre-specified semantic skill labels during both policy and decoder training phases. The left columns present the *BwSC* results, the right columns show the *FwSC* results, while the central columns report the overall combined performance. The best results are indicated in **bold**. In this table, SIL-C utilizes the Type I (PTGM + AA) approach as its foundational architecture.

Kitchen : *Emergent SIL* Seq. 1 (1-2-3-4)

Baselines			*Initial*	*BwSC* (*Initial*)		Overall performance		*FwSC* (*Synced*)		*Final*
Type	Skill Interface	SIL Algo.	FWT (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	FWT (%)
I	Skill Segments [62, 10]	ER	23.5±3.8	-10.8±8.1	15.5±2.3	3.6±7.0	26.6±2.8	17.9±6.5	37.0±3.0	44.5±7.0
	(BUDS)	AA	21.9±4.1	1.3±4.4	22.9±4.0	15.7±3.5	35.3±2.9	30.1±3.9	44.4±2.6	57.1±7.5
	Subgoal Bins [60]	ER	46.8±2.5	-19.7±4.4	32.1±2.9	-5.0±2.8	42.5±1.4	9.7±3.3	54.0±1.8	57.7±3.6
	(PTGM)	AA	45.0±3.1	2.6±0.9	46.9±3.7	15.4±1.7	58.2±1.7	28.1±4.1	66.1±0.6	83.6±3.1
II	Skill-prototypes [11]*	AA	55.1±1.8	0.5±3.1	55.5±2.7	0.5±2.8	55.7±2.7	0.9±2.2	55.8±2.4	56.9±4.9
	Instructions [12]*	ER	54.0±2.4	18.5±4.0	**67.8**±1.1	19.1±2.7	70.3±1.0	19.7±2.0	68.7±1.9	77.4±2.8
III	SIL-C	AA	52.9±2.9	18.6±1.2	66.8±2.6	22.0±2.4	**71.8**±1.3	25.4±4.0	**71.9**±0.9	87.2±3.2
-	Subgoal Bins [60]	Joint	-	-	-	-	-	-	-	86.9±2.2

Kitchen : *Emergent SIL* Seq. 2 (4-3-2-1)

Baselines			*Initial*	*BwSC* (*Initial*)		Overall performance		*FwSC* (*Synced*)		*Final*
Type	Skill Interface	SIL Algo.	FWT (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	FWT (%)
I	Skill Segments [62, 10]	ER	17.9±1.3	-4.4±2.0	14.6±1.1	9.9±2.3	26.4±1.7	24.2±2.8	36.1±1.9	42.1±4.5
	(BUDS)	AA	19.5±1.7	-0.2±2.1	19.4±0.9	14.0±2.2	31.5±1.3	28.2±2.4	40.7±1.7	55.2±4.5
	Subgoal Bins [60]	ER	26.1±1.2	-10.3±2.5	18.4±1.2	9.3±1.3	34.1±0.3	28.8±1.1	47.7±1.0	62.8±3.5
	(PTGM)	AA	25.2±2.2	0.6±1.7	25.6±2.5	19.7±1.0	42.1±1.6	38.9±2.5	54.3±1.8	75.5±2.7
II	Skill-prototypes [11]*	AA	31.9±0.2	-3.1±0.2	29.5±0.3	13.7±1.1	43.6±1.1	30.5±2.0	54.7±1.6	60.5±6.3
	Instructions [12]*	ER	29.8±1.5	-5.3±2.7	25.8±1.2	14.7±2.3	42.3±0.6	34.6±2.5	55.7±0.7	75.7±1.2
III	SIL-C	AA	25.2±2.0	28.9±1.6	**46.9**±3.0	35.7±1.0	**55.8**±1.8	42.5±2.4	**57.0**±1.2	79.3±2.0
-	Subgoal Bins [60]	Joint	-	-	-	-	-	-	-	86.9±2.2

Kitchen : *Emergent SIL* Seq. 3 (3-4-1-2)

Baselines			*Initial*	*BwSC* (*Initial*)		Overall performance		*FwSC* (*Synced*)		*Final*
Type	Skill Interface	SIL Algo.	FWT (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	FWT (%)
I	Skill Segments [62, 10]	ER	32.5±2.7	-13.2±2.1	22.6±1.3	-2.6±2.8	30.3±0.7	8.0±3.5	38.5±0.6	40.4±3.6
	(BUDS)	AA	31.8±1.2	1.2±2.0	32.6±0.8	11.7±1.7	41.8±1.2	22.3±1.7	48.5±1.5	57.2±2.2
	Subgoal Bins [60]	ER	29.7±2.0	-6.9±2.2	24.5±0.8	9.8±3.0	38.1±1.3	26.4±4.3	49.5±2.2	65.2±2.7
	(PTGM)	AA	29.6±1.6	1.1±1.5	30.4±0.7	18.3±1.8	45.3±0.2	35.4±2.2	56.2±0.5	73.3±1.6
II	Skill-prototypes [11]*	AA	30.7±1.1	19.7±0.5	45.5±0.9	25.6±0.6	52.7±1.1	31.5±0.9	54.4±1.3	56.7±0.8
	Instructions [12]*	ER	30.1±2.1	22.1±2.6	46.7±1.3	29.7±2.7	55.6±1.6	37.3±3.2	58.1±2.0	79.1±3.7
III	SIL-C	AA	37.2±2.1	26.5±1.6	**57.1**±1.9	28.9±1.7	**62.0**±2.4	31.2±2.7	**60.6**±3.1	79.5±1.3
-	Subgoal Bins [60]	Joint	-	-	-	-	-	-	-	86.9±2.2

Kitchen : *Emergent SIL* Seq. 4 (2-1-4-3)

Baselines			*Initial*	*BwSC* (*Initial*)		Overall performance		*FwSC* (*Synced*)		*Final*
Type	Skill Interface	SIL Algo.	FWT (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	FWT (%)
I	Skill Segments [62, 10]	ER	31.9±1.9	-20.2±3.6	16.7±1.0	-8.1±3.9	24.9±1.6	4.0±4.7	34.8±2.0	46.4±2.5
	(BUDS)	AA	34.0±2.9	-0.1±3.4	33.9±2.1	7.2±3.2	40.2±1.6	14.6±3.2	44.9±1.1	61.0±3.2
	Subgoal Bins [60]	ER	48.3±0.5	-20.3±4.0	33.0±2.8	-5.5±2.5	43.6±2.2	9.4±2.4	55.3±2.2	59.7±4.8
	(PTGM)	AA	46.4±2.2	1.0±2.0	47.1±1.9	11.4±2.1	56.2±0.9	21.8±2.7	62.7±0.3	72.8±1.4
II	Skill-prototypes [11]*	AA	56.0±3.2	-12.1±4.5	46.9±1.4	-9.1±5.3	48.2±2.0	-6.0±6.2	51.5±2.2	51.7±8.0
	Instructions [12]*	ER	57.9±2.9	4.9±2.2	61.6±1.6	9.7±2.8	66.2±1.1	14.5±3.3	68.8±1.0	76.3±5.4
III	SIL-C	AA	52.8±2.4	17.2±1.0	**65.7**±3.0	20.0±0.9	**69.9**±2.3	22.9±2.8	**69.9**±2.4	80.4±1.4
-	Subgoal Bins [60]	Joint	-	-	-	-	-	-	-	86.9±2.2

D Additional Analysis

D.1 Extended Skill and Subtask Space Resolution Analysis

Table 14 reports the full results corresponding to Figure 6, showing the performance of SIL-C (with PTGM and AA) under different skill and subtask space resolutions. Following the setup in Section 6.3, we test both halved and doubled values of $|\bar{\mathcal{Z}}_p|$ and $|\mathcal{G}_\tau|$ relative to the default configuration.

Table 14: Performance under varying skill and subtask clustering configurations in Kitchen *Emergent SIL* scenario. All baselines use the same SIL algorithm AA for the decoder. Analysis with varying $|\mathcal{G}_\tau|$ and $|\bar{\mathcal{Z}}_p|$ in Kitchen *Emergent SIL* scenario. Here, $|\mathcal{G}_\tau|$ denotes the number of subtasks into which expert demonstration data $\mathcal{D}_\tau$ are segmented and $|\bar{\mathcal{Z}}_p|$ indicates the number of skills derived from the datastream $\mathcal{D}_p$ at each phase p.

Kitchen : *Emergent SIL*											
Baselines				*Initial*	*BwSC* (*Initial*)		Overall performance		*FwSC* (*Synced*)		*Final*
Type	Skill Interface	$\lvert\bar{\mathcal{Z}}_p\rvert$	$\lvert\mathcal{G}_\tau\rvert$	FWT (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	FWT (%)
I	Sub-goal Bins [60] (PTGM)	10	-	46.7±2.0	-0.2±2.1	46.2±1.8	10.5±1.4	55.3±2.6	21.1±1.7	62.2±3.5	76.0±4.9
		20	-	45.0±3.1	2.6±0.9	46.9±3.7	15.4±1.7	58.2±1.7	28.1±4.1	66.1±0.6	83.6±3.1
		40	-	52.8±1.6	-0.5±0.2	52.4±1.5	12.9±0.9	63.8±1.2	26.2±1.7	72.4±1.2	86.4±5.4
III	SIL-C (w/ PTGM)	10	10	51.6±2.1	3.9±2.5	54.5±0.9	11.6±1.6	61.5±1.0	19.4±1.7	66.1±2.0	76.8±2.1
			20	49.8±2.2	5.0±2.3	53.5±3.0	14.0±2.5	61.8±2.2	23.0±3.5	67.1±1.6	78.1±2.6
			40	51.7±2.6	4.2±3.5	54.9±3.6	11.3±3.0	61.4±3.0	18.4±2.7	65.5±2.1	74.0±3.1
		20	10	48.6±1.7	16.0±3.3	60.6±1.3	27.8±14.1	67.1±0.7	27.1±0.7	68.9±1.2	81.3±1.8
			20	52.9±2.9	18.6±1.2	66.8±2.6	22.0±2.4	71.8±1.3	25.4±4.0	71.9±0.9	87.2±3.2
			40	52.3±1.6	16.1±1.2	64.6±1.6	20.8±0.7	70.4±1.5	25.6±0.5	71.7±1.6	86.2±3.1
		40	10	55.1±1.5	18.8±2.3	69.2±1.0	21.5±1.7	73.8±0.4	24.2±2.1	73.3±1.6	88.0±2.4
			20	54.0±1.2	18.8±2.2	68.1±0.5	22.5±1.6	73.3±0.7	26.1±1.7	73.6±1.5	88.8±3.4
			40	55.5±1.0	19.7±2.2	70.3±1.8	23.4±1.4	75.5±1.3	27.1±1.4	75.8±1.1	90.5±0.2

First, we analyze Type I (Subgoal Bins). At $|\bar{\mathcal{Z}}_p| = 10$, the skill space resolution is too coarse, leading to limited overall performance (55.3% AUC). The *BwSC* evaluation group shows weak results (46.2%), and the BWT score remains negative. Increasing to $|\bar{\mathcal{Z}}_p| = 20$ improves overall AUC to 58.2%, and *FwSC* evaluation results rise to 66.1%, but BWT still indicates poor transfer. At $|\bar{\mathcal{Z}}_p| = 40$, Type I gains further, with 63.8% AUC and 72.4% in *FwSC* evaluation, yet *BwSC* remains low (52.4%). These results show that while higher skill space resolution helps Type I, static subgoal binning as the interface is unable to leverage newly added skills without retraining the policy, which is reflected by persistently low BWT in the *BwSC* evaluation group.

When $|\bar{\mathcal{Z}}_p| = 10$, Type III shows clear improvements over Type I. It achieves higher Initial FWT (51.6–51.7%) and stronger AUC in the *BwSC* evaluation group (up to 54.9%). Importantly, the BWT score is positive, indicating that SIL-C enables backward transfer even at coarse resolution, which is not observed in Type I.

When $|\bar{\mathcal{Z}}_p| = 20$, Type III shows consistent improvements over Type I. It achieves higher Initial FWT (48.6–52.3%) and stronger results in both the *BwSC* (up to 68.6%) and *FwSC* (up to 71.9%) evaluation groups. Importantly, the BWT score increases significantly, confirming that SIL-C leverages moderate resolution more effectively than static binning.

When $|\bar{\mathcal{Z}}_p| = 40$, Type III achieves the strongest improvements. It achieves higher Initial FWT (up to 55.5%) and consistently high results in both the *BwSC* (70.3%) and *FwSC* (75.8%) evaluation groups. Most notably, the BWT score peaks at 19.7, the best among all configurations. In contrast, Type I also benefits from higher resolution but remains far behind, with 63.8% overall AUC, 72.4% in *FwSC*, 52.4% in *BwSC*, and negative BWT.

In summary, increasing skill resolution improves both overall performance (AUC) and evaluation results across *BwSC* and *FwSC* groups. Moreover, SIL-C (Type III) consistently leverages this advantage to achieve strong skill-policy compatibility, which is not observed in the static subgoal binning baseline (Type I).

D.2 Subtask Space Analysis on Few-shot Imitation Learning

Table 15 expands on Table 2 by replacing the prototype-based representation in the SIL-C subtask space with an element-wise retrieval strategy across 1-5 shot scenarios. Element-wise retrieval consistently yields stronger performance than prototype-based representations.

In the 5-shot setting, for example, it achieves a +4.2% absolute gain in overall AUC (66.6% vs. 62.9%) and +3.3% in final FWT (79.1% vs. 75.8%). This performance gap persists across shot counts, highlighting the benefit of fine-grained memory when the budget allows. Notably, SIL-C supports both representations and can incorporate element-wise memory when additional storage is available.

Table 15: Comparison between Gaussian prototypes (*Prototype*) and simple instance accumulation (*Element*) in the subtask space of the Franka Kitchen *Emergent* skill incremental scenario under varying shot settings (1, 3, 5). Memory per task indicates the number of embedding vectors stored in the subtask space for a single task (e.g., μ or Σ), with relative usage at 100% corresponding to the configuration in Table 1.

Kitchen : *Emergent SIL*										
Ablations			*Initial*	*BwSC* (*Initial*)		Overall performance		*FwSC* (*Synced*)		*Final*
Instance	Shots	Memory per Task	FWT (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	FWT (%)
Prototype	5	160 (100%)	47.4±3.2	11.3±2.4	55.9±4.6	17.5±1.2	62.4±2.7	23.6±4.2	65.1±2.3	75.8±4.4
	3	160 (100%)	44.4±1.8	11.2±2.5	52.8±2.9	14.9±2.2	57.2±1.8	18.6±3.3	58.4±1.2	68.5±4.0
	1	160 (100%)	43.1±5.4	11.9±2.8	52.0±5.3	15.7±3.1	56.5±3.7	19.4±4.8	57.6±2.9	65.7±3.6
Element-wise	5	1000 (625%)	50.3±4.6	13.9±4.3	60.7±6.4	20.6±1.6	66.6±4.4	24.3±5.3	68.5±2.5	79.1±3.4
	3	600 (375%)	48.9±3.0	15.6±4.6	60.6±2.5	19.3±4.2	65.4±2.1	23.0±3.9	66.1±1.3	77.3±3.6
	1	200 (125%)	42.3±3.6	8.3±3.3	48.5±4.3	14.6±2.4	54.8±2.6	21.0±3.8	58.0±1.9	64.6±3.2

D.3 Robustness to Observation Noise

We extend the analysis in Table 3, which compares SIL-C against PTGM+AA across different noise levels with Gaussian noise scaled by factors of $\times 1$ to $\times 5$ in Franka Kitchen [65].

SIL-C maintains compatibility under noise. As shown in Table 3, SIL-C consistently outperforms PTGM+AA across all noise levels and evaluation metrics. Most notably, SIL-C maintains positive backward transfer (BWT) even under extreme noise ($\times 5$: 4.3% vs. -1.2%), while PTGM+AA suffers from catastrophic forgetting with negative BWT. This gap becomes more pronounced in the final phase with approximately 80 skills, where SIL-C achieves 23.1% FWT compared to 15.9% for PTGM+AA under $\times 5$ noise.

Trajectory matching provides noise resilience. The robustness gap stems from fundamental architectural differences. Under noise, high-level policies produce increasingly unreliable subtask selections. The bilateral lazy learning interface in SIL-C handles this through: (i) skill validation that detects out-of-distribution subtasks, and (ii) skill hooking that remaps to appropriate skills based on trajectory similarity. The static skill mapping of PTGM+AA, however, cannot adapt to noise-induced distributional shifts, leading to compounding errors in long-horizon tasks.

Performance scales with skill repertoire size. The advantage of SIL-C grows as training progresses. Under $\times 3$ noise, the performance gap increases from 0% (Initial FWT: 35.6% vs. 35.8%) to 12.4% (Final FWT: 52.9% vs. 40.5%). This scaling effect suggests that skill validation and hooking become more valuable as the skill library expands, opening the possibility for more robust lifelong learning systems that can maintain performance even in challenging real-world like conditions.

D.4 Analysis on Distance Metrics for Instance-based Classification

Table 16: Comparison across distance metrics (Mahalanobis, Euclidean) and sub-cluster selection strategies (Fixed, Auto) in the Franka Kitchen Emergent Scenario. We define 100% as the configuration matching the number and memory usage of sub-clusters per prototype used in Table 1. *Fixed* uses a predetermined number of sub-clusters, while *Auto* determines the number of sub-clusters using the silhouette score. The left columns report initial forward transfer (*Initial*) and backward skill compatibility (*BwSC*), which evaluate existing policies with updated skills. The right columns report forward skill compatibility (*FwSC*), assessing the effectiveness of new policies with learned skills, and final forward transfer (*Final*). The central columns indicate overall performance, combining *BwSC* and *FwSC*.

Kitchen : *Emergent SIL*										
Baselines			*Initial*	*BwSC* (*Initial*)		Overall performance		*FwSC* (*Synced*)		*Final*
Type	Distance Type	Sub-clusters per prototype	FWT (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	FWT (%)
III	Mahalanobis	Fixed $K_{\bar{z}}, K_{\bar{g}} = 4.0$ (100%)	52.9±2.9	18.6±1.2	66.8±2.6	22.0±2.4	71.8±1.3	25.4±4.0	71.9±0.9	87.2±3.2
		Auto $K_{\bar{z}}, K_{\bar{g}} \approx 2.5$ (63.5%)	52.7±3.0	16.6±2.5	65.1±2.0	19.6±2.6	69.5±1.6	22.7±2.9	69.7±1.4	80.6±3.2
	Euclidean	Fixed $K_{\bar{z}}, K_{\bar{g}} = 4.0$ (50%)	52.2±4.7	5.5±2.3	56.3±3.1	14.3±4.7	64.5±0.8	23.2±7.2	69.6±0.7	81.0±3.6
		Fixed $K_{\bar{z}}, K_{\bar{g}} = 8.0$ (100%)	54.0±2.6	5.6±2.8	58.2±1.8	14.4±1.9	66.3±1.7	23.1±1.4	71.3±2.0	82.9±2.8

We ablate the choice of distance metric in the instance-based classifier, comparing Mahalanobis and Euclidean distances with varying memory configurations as shown in Table 16. Memory usage is measured relative to storing 4 sub-clusters with both mean and covariance per prototype (100% baseline).

Mahalanobis distance enables better trajectory matching. Across all configurations, Mahalanobis distance consistently outperforms Euclidean distance. With the baseline configuration, Mahalanobis achieves 66.8% AUC in BwSC while Euclidean only reaches 56.3%, a 10.5% gap. This difference is most pronounced in backward compatibility, where Mahalanobis maintains 18.6% BWT compared to 5.5% for Euclidean, demonstrating superior ability to align evolving skill distributions with existing policies.

Memory-performance trade-off with automatic clustering. When using automatic sub-cluster selection via silhouette score, the Mahalanobis variant reduces memory to 63.5% while maintaining competitive performance (65.1% vs. 66.8% AUC in BwSC). This 3.2% performance drop for 36.5% memory savings presents a viable option for resource-constrained deployments, as the SIL-C still significantly outperforms Euclidean variants even with reduced memory.

Richer skill distribution modeling improves compatibility. The performance gap between distance metrics demonstrates the importance of capturing skill distributions accurately. Euclidean distance with 4 sub-clusters (50% memory) achieves only 56.3% AUC in BwSC, and even doubling to 8 sub-clusters (100% memory) reaches just 58.2% AUC. In contrast, Mahalanobis distance with 4 sub-clusters achieves 66.8% AUC despite using diagonal covariance for efficiency. This 8.6%-10.5% gap shows that modeling variance, even in simplified diagonal form, is more effective than simply adding more isotropic prototypes. The Euclidean approach scales poorly because increasing prototypes cannot compensate for the lack of directional information, while Mahalanobis captures essential trajectory variations that distinguish functionally different skills. This enables more accurate skill validation and hooking, particularly critical for backward compatibility where precise distribution matching determines whether existing policies can leverage new skills.

D.5 Analysis on Instance-based Classifier Thresholding

Figure 10 shows the stability of confidence intervals used for threshold selection in the *Emergent SIL* scenario. Overall, varying the threshold between 80% and 99% results in minimal differences in both the confidence intervals and performance. This indicates that our bounding method does not simply adjust the threshold for skill validation, but instead serves as an effective mechanism for filtering out out-of-distribution cases.

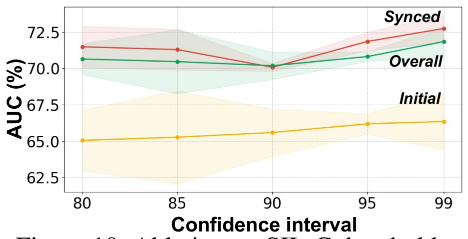


Figure 10: Ablation on SIL-C threshold

D.6 Memory and Time Analysis

Memory. Each instance-based classifier stores Gaussian prototypes in both skill and subtask spaces. The memory cost for skill prototypes is $dim \times |\mathcal{Z}_p| \times K_z \times 2$ float values, and for subtask prototypes it is $dim \times |\mathcal{G}_\tau| \times K_g \times 2$. Here, dim is the dimension of the state or subgoal embedding. Each (μ, Σ) pair is stored using a diagonal covariance matrix. K_z and K_g denote the average number of sub-clusters per skill and subtask, respectively.

In the Kitchen *Emergent* and *Explicit* skill incremental settings with $dim = 60$, skill prototypes occupy 150KB and subtask prototypes occupy 37.5KB, as shown in Table 1. In the Meta-World benchmark with $dim = 140$, the memory usage increases to 350KB for skill prototypes and 87.5KB for subtask prototypes.

Time. Each Mahalanobis distance computation with diagonal covariance requires $4 \times dim$ floating point operations: dim subtractions, dim squarings, dim divisions, $dim - 1$ additions, and one square root. For $dim = 60$, the total is 240 FLOPs per (μ, Σ) pair.

In the best case, a proposed z_h is directly accepted as the skill z_l during the skill validation process using a selected subgoal from the subtask space. This process requires $(|\mathcal{G}_\tau| \times K_g + 1) \times 240$ FLOPs. In the worst case, z_h is rejected during skill validation, and skill hooking requires selecting candidate skills $\mathcal{Z}'$ and scoring them to choose the appropriate skill z_l. The total computational cost is up to $\{(|\mathcal{G}_\tau| \times K_g + 1) + (2 \times |\mathcal{Z}_p| \times K_z + 1)\} \times 240$ FLOPs.

Table 17: Mean and variance (ms) of evaluation function timing in the Kitchen *Emergent* Skill Incremental scenario. *kbls* denotes the task with the following sequence: `move kettle, turn on bottom burner, light switch, open slide cabinet`, and *btls* denotes: `turn on bottom burner, turn on top burner, light switch, open slide cabinet`.

Baselines		***kbls-Initial***		***btls-Initial***		***kbls-Final***		***btls-Final***	
Type	**Method**	**Mean**	**Var**	**Mean**	**Var**	**Mean**	**Var**	**Mean**	**Var**
I	PTGM + AA	28.089	2.042	28.117	1.784	27.693	1.978	28.052	2.191
III	SIL-C + (PTGM + AA)	29.042	2.341	27.782	2.324	28.787	2.732	28.538	2.320

Table 17 reports the inference time (mean and variance) of the hierarchical model, consisting of the policy, interface, and skill decoder, in the Kitchen *Emergent* Skill Incremental scenario. All timings were measured on an AMD Ryzen 9 7950X3D CPU with a single NVIDIA RTX 4090 GPU, running Ubuntu 22.04, CUDA 12.4, and driver version 550.144.03. The *Initial* phase ($p = 1$) corresponds to the evaluation after 20 skill prototypes were generated, while the *Final* phase ($p = 4$) reflects evaluation after the final phase with a total of 80 skills. Compared to Type I, the mean inference time of Type III increased by 0.953 ms and 1.094 ms in the *kbls-Initial* and *kbls-Final* , respectively. In the *btls-Initial*, the mean time decreased by 0.335 ms, while in the *btls-Final*, it increased by 0.486 ms. While a greater number of skills generally leads to longer retrieval time, the additional overhead was minimal compared to overall policy and skill decoder inference time, and variation due to runtime system state was more prominent in practice.

NeurIPS Paper Checklist

- **Claims**

  Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

  Answer: [Yes]

  Justification: The abstract and introduction explicitly state the motivation and objective of SIL-C and outline three core contributions corresponding to the methods and experiments.

  Guidelines:

  - The answer NA means that the abstract and introduction do not include the claims made in the paper.
  - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
  - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
  - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

- **Limitations**

  Question: Does the paper discuss the limitations of the work performed by the authors?

  Answer: [Yes]

  Justification: We outline future work that aims to address current limitations by incorporating external feedback, particularly in light of the trade-off observed between generalizability and compatibility.

  Guidelines:

  - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
  - The authors are encouraged to create a separate "Limitations" section in their paper.
  - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
  - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
  - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
  - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
  - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
  - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

- **Theory assumptions and proofs**

  Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: We support our method by showing experimental results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

- **Experimental result reproducibility**

  Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

  Answer: [Yes]

  Justification: Implementation details, scenarios, metrics and datasets are described in Sections 5-6 and Appendix A-D; also abstract provide the Git repository.

  Guidelines:

  - The answer NA means that the paper does not include experiments.
  - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
  - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
  - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
  - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
    - If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
    - If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
    - If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
    - We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

- **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: A public anonymized Git repository is provided in abstract which contains code and instructions. Experimental settings are also discussed throughout section 5-6 and appendix.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

- **Experimental setting/details**

  Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

  Answer: [Yes]

  Justification: Section 5 and Appendix A give datasets, hyper-parameters, continual learning methods and evaluation metrics.

  Guidelines:

  - The answer NA means that the paper does not include experiments.
  - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
  - The full details can be provided either with the code, in appendix, or as supplemental material.

- **Experiment statistical significance**

  Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

  Answer: [Yes]

  Justification: We report our results with variance, using at least four seeds for each experiment.

  Guidelines:

  - The answer NA means that the paper does not include experiments.
  - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

  - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
  - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
  - The assumptions made should be given (e.g., Normally distributed errors).
  - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
  - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
  - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
  - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

- **Experiments compute resources**

  Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

  Answer: [Yes]

  Justification: We provide them in appendix

  Guidelines:

  - The answer NA means that the paper does not include experiments.
  - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
  - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
  - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

- **Code of ethics**

  Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

  Answer: [Yes]

  Justification: We adhere to the NeurIPS Code of Ethics in every respect.

  Guidelines:

  - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
  - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
  - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

- **Broader impacts**

  Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

  Answer: [Yes]

  Justification: We discuss the the potential impacts in the conclusion section.

  Guidelines:

  - The answer NA means that there is no societal impact of the work performed.
  - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

- **Safeguards**

  Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

  Answer: [NA]

  Justification: This work is not expected to pose any significant risk of negative societal impact thus explicit safeguards for responsible data or model release were considered unnecessary.

  Guidelines:

  - The answer NA means that the paper poses no such risks.
  - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
  - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
  - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

- **Licenses for existing assets**

  Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

  Answer: [Yes]

  Justification: All baselines and datasets are cited in the References section. Additional information, including license and version details, will be provided in the appendix.

  Guidelines:

  - The answer NA means that the paper does not use existing assets.
  - The authors should cite the original paper that produced the code package or dataset.
  - The authors should state which version of the asset is used and, if possible, include a URL.
  - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
  - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

- **New assets**

  Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

  Answer: [NA]

  Justification: No new assets were introduced in this paper.

  Guidelines:

  - The answer NA means that the paper does not release new assets.
  - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
  - The paper should discuss whether and how consent was obtained from people whose asset is used.
  - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

- **Crowdsourcing and research with human subjects**

  Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

  Answer: [NA]

  Justification: Not applicable to this paper.

  Guidelines:

  - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
  - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
  - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

- **Institutional review board (IRB) approvals or equivalent for research with human subjects**

  Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

  Answer: [NA]

  Justification: Not applicable to this paper.

  Guidelines:

  - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
  - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

- **Declaration of LLM usage**

  Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

  Answer: [NA]

  Justification: Not applicable to this paper.

  Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.