

488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507

---

# Appendix

---

## Contents

<b>A</b>	<b>SIL-C Implementation</b>	<b>15</b>
A.1	Notations . . . . .	15
A.2	Algorithms . . . . .	16
A.3	Hyperparameters . . . . .	17
<b>B</b>	<b>Experiments Settings</b>	<b>17</b>
B.1	SIL Scenario . . . . .	17
B.2	Baseline Details . . . . .	19
B.3	Metrics . . . . .	21
B.4	Training Details . . . . .	21
B.5	Evaluation Details . . . . .	22
<b>C</b>	<b>Additional Results</b>	<b>23</b>
C.1	Skill-Poily Compatibility : Overview and Ordering Effects . . . . .	23
C.2	Prototype Scalability Analysis . . . . .	24
C.3	Distance Analysis . . . . .	25
C.4	SIL-C Subtask Space Analysis . . . . .	26
C.5	Memory and Time Analysis . . . . .	26

## A SIL-C Implementation

This section details key notations in A.1 and operational steps in A.2 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 section defines key symbols and terms used throughout the paper.

Table 3: Notation used in the paper

Symbol	Meaning	Symbol	Meaning
<b>MDP fundamentals</b>			
$\mathcal{M}$	$(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mu_0, \gamma)$	$\mathcal{S}$	State space
$\mathcal{A}$	Action space	$\mathcal{P}$	Transition
$\mathcal{R}$	Reward function	$\mu_0$	Initial-state distribution
$\gamma$	Discount factor ( $0 < \gamma < 1$ )	$s$	State
$a$	Action	$r, R$	Step reward, episode return
$\pi$	Policy	$\tau$	Task identifier
<b>Hierarchical policy</b>			
$\pi_h$	High-level policy	$\pi_l$	Low-level skill decoder
$z_h$	High-level subtask	$z_l$	Low-level skill
$\theta_h$	High-level policy parameters	$\theta_l$	Low-level skill decoder parameters
$\psi$	Mapping $z_h \rightarrow z_l$	$\mathcal{Z}$	Subtask/skill space
<b>Skill Incremental Learning (SIL)</b>			
$p$	SIL phase index	$P$	Total number of SIL phases
$\{\mathcal{D}_p\}_{p=1}^P$	Datastream	$\mathcal{T}_p$	Evaluation tasks at phase $p$
$\mathcal{Z}_p$	Subtask/skill index set at phase $p$	$\mathcal{D}_\tau$	Expert demos for task $\tau$
$\pi_h^\tau$	High-level policy for task $\tau$	$\pi_l^{(p)}$	Skill decoder at phase $p$
$\theta_h^\tau$	Parameters of $\pi_h$ for task $\tau$	$\theta_l^{(p)}$	Parameters of $\pi_l$ at phase $p$
<b>Lazy Learning Skill Interface</b>			
$g$	Subgoal state	$m$	Steps ahead to define a subgoal state
$\Psi_i^s(s)$	Skill-side classifier: skill from state $s$	$\mathcal{X}_i^{s,(p)}$	Skill memory (state-based prototypes)
$\Psi_i^g(g)$	Skill-side classifier: skill from subgoal state $g$	$\mathcal{X}_i^{g,(p)}$	Skill memory (subgoal-based prototypes)
$\Psi_h^s(s)$	Task-side classifier: subgoal prediction from state	$\mathcal{X}_h^{s,\tau}$	Task-specific memory (subtask prototypes)
$f(\cdot)$	Interface decision rule	$\mathcal{Z}'$	Set of candidate skills compatible with subgoal
$\delta_c$	Distance threshold for class $c$	$d_c(x)$	Mahalanobis distance to class prototype
$\chi_c$	Prototype for class $c$	$\mu_{c,k}, \Sigma_{c,k}$	Mean and covariance of $k$ -th Gaussian in $\chi_c$
$K_c$	Number of components(modes) in prototype $\chi_c$	$x$	Query instance (e.g., $s$ or $g$ )
<b>Prototype Construction</b>			
$\mathcal{E}_{\bar{z}}$	Clustered skill group	$\bar{\mathcal{Z}}_p$	Index set of discovered skills at phase $p$
$\mathcal{H}_{\bar{z},k}^s, \mathcal{H}_{\bar{z},k}^g$	Sub-clusters for skill $\bar{z}$	$K_{\bar{z}}$	Num. sub-clusters for skill $\bar{z}$
$\chi_{\bar{z}}, \chi_{\bar{z}}^g$	Prototypes for skill $\bar{z}$ (state/subgoal)	$\mu_{\bar{z},k}, \Sigma_{\bar{z},k}$	Mean, covariance of skill $\bar{z}$ 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 $\bar{g}$	$K_{\bar{g}}$	Num. sub-clusters for subtask $\bar{g}$
$\chi_{\bar{g}}^s$	Subtask prototype (state)	$\mu_{\bar{g},k}, \Sigma_{\bar{g},k}$	Mean, covariance of subtask $\bar{g}$ cluster

515 We present the algorithms for skill space updates with skill decoder learning (Section 4.2), subtask  
 516 space updates with policy learning (Section 4.2), and the inference process (Section 4.3).

---

**Algorithm 1** SIL-C Skill Incremental Learning with Interface
 

---

**Require:** Phase index  $p$ , streamed dataset  $\mathcal{D}_p$ , previous decoder  $\pi_l^{(p-1)}$ ,  
 memories  $\mathcal{X}_l^{s,(p-1)}, \mathcal{X}_l^{g,(p-1)}$   
**Ensure:** Updated decoder  $\pi_l^{(p)}$ , memories  $\mathcal{X}_l^{s,(p)}, \mathcal{X}_l^{g,(p)}$   
 1: // (a) Creating new skill prototypes (Subsection 4.2: Skill Space Update)  
 2: **Skill clustering** segment  $\mathcal{D}_p$  into  $\{\mathcal{E}_{\bar{z}}\}_{\bar{z} \in \bar{\mathcal{Z}}_p}$  using Eq. (6)  
 3: Initialize temporary sets  $\bar{\mathcal{X}}_l^{s,(p)} \leftarrow \emptyset, \bar{\mathcal{X}}_l^{g,(p)} \leftarrow \emptyset$   
 4: **for all**  $\bar{z} \in \bar{\mathcal{Z}}_p$  **do**  
 5:   **Sub-clustering** apply K-means to  $\mathcal{E}_{\bar{z}}$  to obtain  $\{\mathcal{H}_{\bar{z},k}^s, \mathcal{H}_{\bar{z},k}^g\}_{k=1}^{K_{\bar{z}}}$   
 6:   **Prototype creation** compute  $\{\mu_{\bar{z},k}, \Sigma_{\bar{z},k}\}_{k=1}^{K_{\bar{z}}}$  and form  $\chi_{\bar{z}}^s, \chi_{\bar{z}}^g$   
 7:   **Add prototypes:**  $\bar{\mathcal{X}}_l^{s,(p)} \leftarrow \bar{\mathcal{X}}_l^{s,(p)} \cup \{\chi_{\bar{z}}^s\}, \bar{\mathcal{X}}_l^{g,(p)} \leftarrow \bar{\mathcal{X}}_l^{g,(p)} \cup \{\chi_{\bar{z}}^g\}$   
 8: **end for**  
 9: **Memory append**  $\mathcal{X}_l^{s,(p)} \leftarrow \mathcal{X}_l^{s,(p-1)} \cup \bar{\mathcal{X}}_l^{s,(p)}, \mathcal{X}_l^{g,(p)} \leftarrow \mathcal{X}_l^{g,(p-1)} \cup \bar{\mathcal{X}}_l^{g,(p)}$   
 10: // (b) Updating skill decoder and Accumulating skill index  
 11: **Decoder update** train  $\pi_l$  with  $A_l$  on  $\mathcal{D}_p$  to obtain  $\pi_l^{(p)}$   
 12: **Index growth**  $\mathcal{Z}_p \leftarrow \mathcal{Z}_{p-1} \cup \bar{\mathcal{Z}}_p$

---



---

**Algorithm 2** SIL-C Policy Training with Interface
 

---

**Require:** Task  $\tau$ , expert demonstrations  $\mathcal{D}_\tau$ , current decoder  $\pi_l^{(p)}$ , global skill index set  $\mathcal{Z}_p$   
**Ensure:** High-level policy  $\pi_h^\tau$ , task memory  $\mathcal{X}_h^{s,\tau}$   
 1: // (a) Creating subtask prototypes (Subsection 4.2: Subtask Space Update)  
 2: **Subtask clustering** segment  $\mathcal{D}_\tau$  into  $\{\mathcal{E}_{\bar{g}}\}_{\bar{g} \in \mathcal{G}_\tau}$  (Eq. (7))  
 3: **for all**  $\bar{g} \in \mathcal{G}_\tau$  **do**  
 4:   **Sub-clustering** apply K-means to  $\mathcal{E}_{\bar{g}}$  to obtain  $\{\mathcal{H}_{\bar{g},k}^s\}_{k=1}^{K_{\bar{g}}}$   
 5:   **Prototype creation** compute  $\{\mu_{\bar{g},k}, \Sigma_{\bar{g},k}\}_{k=1}^{K_{\bar{g}}}$  and form  $\chi_{\bar{g}}^s$   
 6:   **Add prototypes:**  $\mathcal{X}_h^{s,\tau} \leftarrow \mathcal{X}_h^{s,\tau} \cup \{\chi_{\bar{g}}^s\}$   
 7: **end for**  
 8: // (b) Assigning labels and optimizing high-level policy (Subsection 4.3: Policy Learning)  
 9: **for all** transition  $(s, a^*) \in \mathcal{D}_\tau$  **do**  
 10:   **Label assignment**  $z_h^* \leftarrow \arg \min_{z \in \mathcal{Z}_p} \|\pi_l^{(p)}(a | s, z) - a^*\|_2$  (Eq. (8))  
 11: **end for**  
 12: **Policy optimization** update  $\pi_h^\tau$  by minimizing  $\mathbb{E}_{(s, z_h^*)} [-\log \pi_h^\tau(z_h^* | s)]$  using the prior

---

---

**Algorithm 3** SIL-C Evaluation with Interface

---

**Require:** Decoder  $\pi_l^{(p)}$ , policy  $\pi_h^\tau$ , interface modules  $\Psi_l^{s,(p)}$ ,  $\Psi_l^{g,(p)}$ ,  $\Psi_h^{s,\tau}$ , initial state  $s$

- 1: Initialize accumulated return  $R \leftarrow 0$
- 2: **while** episode not terminated **do**
- 3:   **// (a) Policy inference (Subsection 4.1: Bilateral Modules)**
- 4:    $z_h \sim \pi_h^\tau(z \mid s)$  *// sample subtask*
- 5:   **// (b) Interface inference (Subsection 4.3: Policy inference via Interface)**
- 6:    $g \leftarrow \Psi_h^{s,\tau}(s)$  *// sample subgoal state*
- 7:   **if**  $d_{z_h}(g) \leq \delta_{z_h}$  **then**
- 8:      $z_l \leftarrow z_h$  *// accept subtask as skill*
- 9:   **else**
- 10:     $\mathcal{Z}' \leftarrow \{z' \in \mathcal{Z}_p \mid \Psi_l^{g,(p)}(g) \leq \delta_{z'}\} \cup \{z_h\}$  *// find candidate skills*
- 11:     $z_l \leftarrow \arg \min_{z' \in \mathcal{Z}'} \Psi_l^{s,(p)}(s)$  *// skill hooking*
- 12:   **end if**
- 13:   **// (b) Executing skill and accumulating rewards**
- 14:    $a \sim \pi_l^{(p)}(a \mid s, z_l)$  *// sample action*
- 15:   Execute  $a$ , observe reward  $r$  and next state  $s$ , update  $R \leftarrow R + r$
- 16: **end while**

---

### 517 A.3 Hyperparameters

518 We apply a consistent set of hyperparameters to update each space in the SIL-C interface for both the  
519 Kitchen and Meta-World environments. The configuration, used in Table 1, is summarized in Table 4.

Table 4: 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

## 520 B Experiments Settings

### 521 B.1 SIL Scenario

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

#### 524 B.1.1 Environments

525 **Franka Kitchen** [64] is a long-horizon manipulation environment where a robot must complete  
526 multi-stage tasks by interacting with various kitchen objects such as a microwave, kettle,  
527 burners, light switches, and cabinet doors. We use the kitchen-mixed-v0 dataset  
528 for experiment. Each demonstration consists solely of state-action transitions, without any re-  
529 ward signals or task labels. The state space is 60-dimensional, encompassing both the robot  
530 arm state and the states of the manipulated objects. The action space is 9-dimensional, corre-  
531 sponding to the 9 degrees of freedom (DoF) of the Franka robotic arm. Each task requires the  
532 robot to complete a sequence of 4 subtasks(stage) selected from a fixed set of 7 predefined sub-  
533 tasks: open microwave, move kettle, turn on top burner, turn on bottom burner,  
534 light switch, open hinge cabinet, open slide cabinet. A reward of 1.00 is given for  
535 each subtask successfully completed in the correct order, yielding a maximum of 4.00 when all  
536 subtasks are completed. For evaluation, we normalize the reward such that the maximum score is  
537 100.

538 **Multi-stage Meta-World** extends the original Meta-World benchmark, which includes 50 diverse  
539 single-step robot tasks, by composing multiple tasks into sequential stages within a single episode.  
540 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. We use the Easy dataset introduced by [9], 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.1.2 Datastream Types : Emergent and Explicit Skill Incremental

Table 5: Kitchen: *Emergent* Skill Incremental Settings

Kitchen : <i>Emergent</i> Skill Incremental				
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
2	microwave	bottom burner	top burner	hinge cabinet
	microwave	kettle	bottom burner	hinge cabinet
	microwave	kettle	top burner	hinge cabinet
	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

Table 6: Meta-World: *Emergent* Skill Incremental Settings

Meta-World: <i>Emergent</i> Skill Incremental Settings				
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
3	button	puck	drawer	door
	button	puck	door	drawer
	button	drawer	puck	door
	button	drawer	door	puck
	button	door	puck	drawer
4	door	puck	drawer	button
	door	puck	button	drawer
	door	drawer	puck	button
	door	drawer	button	puck
	door	button	puck	drawer

Table 7: Kitchen: *Explicit* Skill Incremental Settings

Kitchen: <i>Explicit</i> Skill Incremental	
Phase	Skill Datasets
1	microwave
2	kettle / bottom burner
3	top burner / light switch
4	slide cabinet / hinge cabinet

Table 8: Meta-World: *Explicit* Skill Incremental Settings

Meta-World: <i>Explicit</i> Skill Settings	
Phase	Skill Datasets
1	puck
2	drawer
3	button
4	door

**Emergent Skill Incremental.** In the *Emergent* Skill Incremental 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 5. 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 6.

**Explicit Skill Incremental.** In the *Explicit Skill Incremental* 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 7. 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 8.

### B.1.3 Evaluation Groups for SIL

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 policy decoders, one for each evaluation task. Across both environments, we evaluate all 24 tasks listed in Table 5 and Table 6 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 $_{\tau}$  and FwSC $_{\tau}$  of task  $\tau$  are defined as:

$$\text{BwSC}_{\tau} = [(\pi_h^{\tau,(1)}, \pi_l^{(1)}), (\pi_h^{\tau,(1)}, \pi_l^{(2)}), \dots, (\pi_h^{\tau,(1)}, \pi_l^{(P)})] \quad (10)$$

$$\text{FwSC}_{\tau} = [(\pi_h^{\tau,(1)}, \pi_l^{(1)}), (\pi_h^{\tau,(2)}, \pi_l^{(2)}), \dots, (\pi_h^{\tau,(P)}, \pi_l^{(P)})] \quad (11)$$

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  $p > 1$ ,  $\pi_h^{\tau,(p)}$  refers to the updated high-level policy synchronized with the corresponding skill decoder  $\pi_l^{(p)}$  at phase  $p$ .

## B.2 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** [61, 11]: BUDS [61] segments trajectories into fixed-length intervals and merges them bottom-up based on trajectory similarity, and was later adopted as the foundation in LOTUS [11] 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** [59]: 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

610 from the PTGM paper for the Kitchen domain. Each phase produces a total of 20 skill  
611 clusters.

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

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

- 617 • **Fine-Tuning (FT)**: The simplest approach. The skill decoder is incrementally trained on  
618 each new skill without access to prior data, no memory or replay is used.
- 619 • **Experience Replay (ER)** [68]: In our implementation, a replay buffer stores 10% of the  
620 data from each previous phase. During training in the next phase, these stored samples are  
621 interleaved with new data at a 1:1 sampling ratio to mitigate forgetting.
- 622 • **Adapter Append (AA)** [38]: In the first phase, we pre-train the entire model. Starting from  
623 the second phase, we freeze the pre-trained model and train a separate LoRA adapter [75]  
624 for each new phase, using rank 4 for the *Emergent* scenario and rank 16 for the *Explicit*  
625 scenario. During evaluation, we use the adapter corresponding to the phase in which the skill  
626 was introduced. This strategy, introduced in TAIL [38], preserves skill-policy compatibility  
627 by preventing forgetting in both the base model and its adapted parameters. However, it  
628 limits the ability to transfer or incorporate newly acquired skills across different phases.

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

- 632 • **Skill-prototypes + (AA)** [9]: Performs skill retrieval using a prototype memory, where each  
633 prototype is linked with adapter parameters and indexed by persistent semantic skill labels.  
634 We set the number of skill prototype bases to 50. Following the original setup, tasks are  
635 learned using semantic subgoal labels provided in a fixed sequence.
- 636 • **Instructions + (ER+AA)** [13]: Defines and learns skills incrementally from semantic  
637 representations. We adopt its model expansion strategy combined with trajectory-based  
638 temporal replay to support skill incremental learning.

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

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

646 **High-level policy and skill decoder implementation.** For all baselines, we implement the high-level  
647 policy as a four-layer MLP that acts as a classifier for sub-task prediction. To enable behavior cloning  
648 in SIL-C, we generate supervision signals using Eq. (8) and train the classifier with cross-entropy  
649 loss.

650 The skill decoder, used across all baselines, is a goal-conditioned policy consisting of two components.  
651 The first maps each skill to a hashed embedding. Following the skill decoder architecture proposed in  
652 [59], we compute each skill embedding as the subgoal state closest to the centroid of the subgoal  
653 states comprising that skill. We then feed this subgoal state, along with the current state, into a  
654 conditional denoising diffusion model with four conditioned denoising blocks. The model denoises  
655 from Gaussian noise to reconstruct the corresponding action.

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

Table 10: Default skill decoder configuration

Hyperparameter	Value
Diffusion Model	DDPM [69]
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	$2 \times 10^{-5}$
$\beta_1$	0.9

Table 9: Default policy configuration

Hyperparameter	Value
Model	MLP
Hidden size	512
Hidden layers	4
Dropout	0.1
State input dim	60
Optimizer	Adam
Learning rate	$1 \times 10^{-4}$
$\beta_1$	0.9

### 659 B.3 Metrics

660 For each scenario, we evaluate the *FwSC* and *BwSC* groups using three metrics: Forward Transfer  
661 (FWT), Backward Transfer (BWT), and Area Under the Curve (AUC). *Overall performance* is  
662 reported by computing each metric over the union of both groups. Let  $r^{\tau,(p)}$  denote the normalized  
663 reward for task  $\tau$  at phase  $p$ , evaluated using the corresponding high-level policy and skill decoder. For  
664 example,  $r^{\tau,(p)} = (\pi_h^{\tau,(1)}, \pi_l^{(p)})$  for the *FwSC* group, and  $r^{\tau,(p)} = (\pi_h^{\tau,(p)}, \pi_l^{(p)})$  for the *BwSC* group.  
665 This score reflects task performance under the given high-level and low-level policy configuration.  
666 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  
667  $p \in 1, \dots, P$ .

668 **Forward Transfer (FWT)** measures the effectiveness of newly learned skills when applied to  
669 downstream task  $\tau$  at phase  $p$ , while keeping the high-level policy fixed from initial phase  $p = 1$ :

$$\text{FWT}^{\tau,(p)} = r^{\tau,(p)} \quad (12)$$

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

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

$$\text{BWT}^{\tau} = \frac{1}{P-1} \sum_{p=2}^P \left( r^{\tau,(p)} - r^{\tau,(1)} \right) \quad (13)$$

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

$$\text{AUC}^{\tau} = \frac{1}{P} \sum_{p=1}^P r^{\tau,(p)} \quad (14)$$

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

### 678 B.4 Training Details

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

- 680 • AMD Ryzen 9 7950X3D 16-Core Processor with a single RTX 4090 GPU. OS: Ubuntu  
681 22.04, CUDA Version: 12.4, Driver Version: 550.144.03

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

686 **Training Framework Details** We implemented our framework using JAX [76] to efficiently ac-  
687 celerate training and handle the continual phase’s alternating training and evaluation process. All  
688 software dependencies are included with the released code for full reproducibility. The versions of  
689 core libraries are:

- 690 • `jax`: 0.4.34
- 691 • `jaxlib`: 0.4.34
- 692 • `flax`: 0.10.2
- 693 • `optax`: 0.1.9

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

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

#### 701 **Total Compute Usage**

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

#### 709 **B.5 Evaluation Details**

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

## C Additional Results

### C.1 Skill-Poiley Compatibility : Overview and Ordering Effects

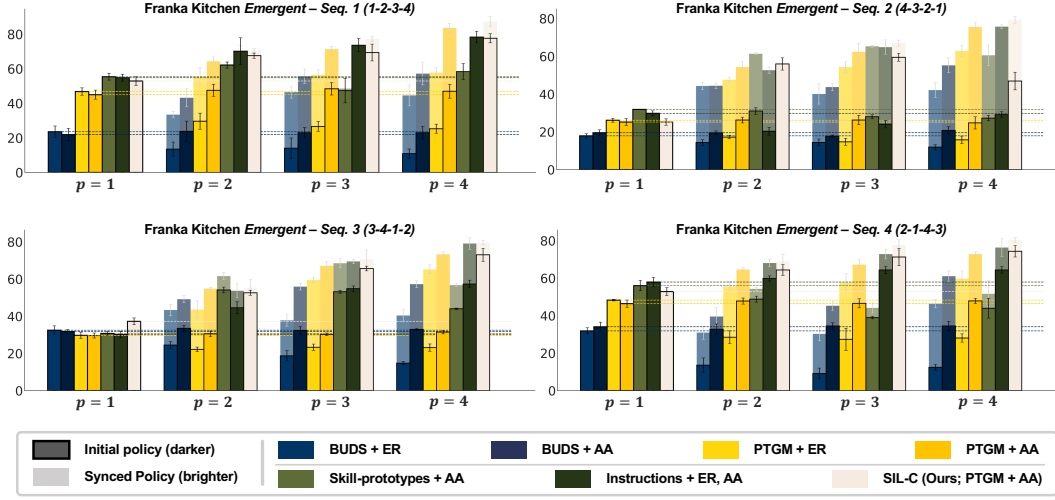


Figure 7: Phase-wise normalized rewards across four Kitchen *Emergent* Skill Incremental datastreams, each corresponding to a different permutation of skill execution order. Each bar group shows performance for different methods under both initial and retrained (Synced) policies.

Figure 7 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 11. These experiments assess both backward skill compatibility (*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 5. 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 11: 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 **in bold**. In this table, SIL-C utilizes the Type I (PTGM + AA) approach as its foundational architecture.

Kitchen : Emergent Skill Incremental 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 [61, 11] (BUDS)	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
	Sub-goal Bins [59] (PTGM)	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 [9]*	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 [13]*	ER+AA	54.0±2.4	18.5±4.0	<b>67.8±1.1</b>	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	<b>71.8±1.3</b>	25.4±4.0	<b>71.9±0.9</b>	87.2±3.2
-	Sub-goal Bins [59]	Joint	-	-	-	-	-	-	-	86.9±2.2
Kitchen : Emergent Skill Incremental 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 [61, 11] (BUDS)	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
		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
	Sub-goal Bins [59] (PTGM)	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
		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 [9]*	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 [13]*	ER+AA	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	<b>46.9±3.0</b>	35.7±1.0	<b>55.8±1.8</b>	42.5±2.4	<b>57.0±1.2</b>	79.3±2.0
-	Sub-goal Bins [59]	Joint	-	-	-	-	-	-	-	86.9±2.2
Kitchen : Emergent Skill Incremental 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 [61, 11] (BUDS)	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
		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
	Sub-goal Bins [59] (PTGM)	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
		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 [9]*	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 [13]*	ER+AA	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	<b>57.1±1.9</b>	28.9±1.7	<b>62.0±2.4</b>	31.2±2.7	<b>60.6±3.1</b>	79.5±1.3
-	Sub-goal Bins [59]	Joint	-	-	-	-	-	-	-	86.9±2.2
Kitchen : Emergent Skill Incremental 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 [61, 11] (BUDS)	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
		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
	Sub-goal Bins [59] (PTGM)	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
		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 [9]*	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 [13]*	ER+AA	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	<b>65.7±3.0</b>	20.0±0.9	<b>69.9±2.3</b>	22.9±2.8	<b>69.9±2.4</b>	80.4±1.4
-	Sub-goal Bins [59]	Joint	-	-	-	-	-	-	-	86.9±2.2

## 738 C.2 Prototype Scalability Analysis

739 Figure 8 presents an analysis of AUC performance in the Kitchen *Emergent* skill incremental scenario,  
740 comparing different numbers of segments used to construct the skill and subtask prototypes in SIL-C.  
741 Overall, setting  $|\bar{\mathcal{Z}}_p| = 40$  yields the highest AUC across all evaluation groups within a single SIL  
742 stream. Additionally, when  $|\bar{\mathcal{Z}}_p| = 20$ , reducing  $|\bar{\mathcal{Q}}_p|$  to 10 leads to a drop of more than 10% in AUC

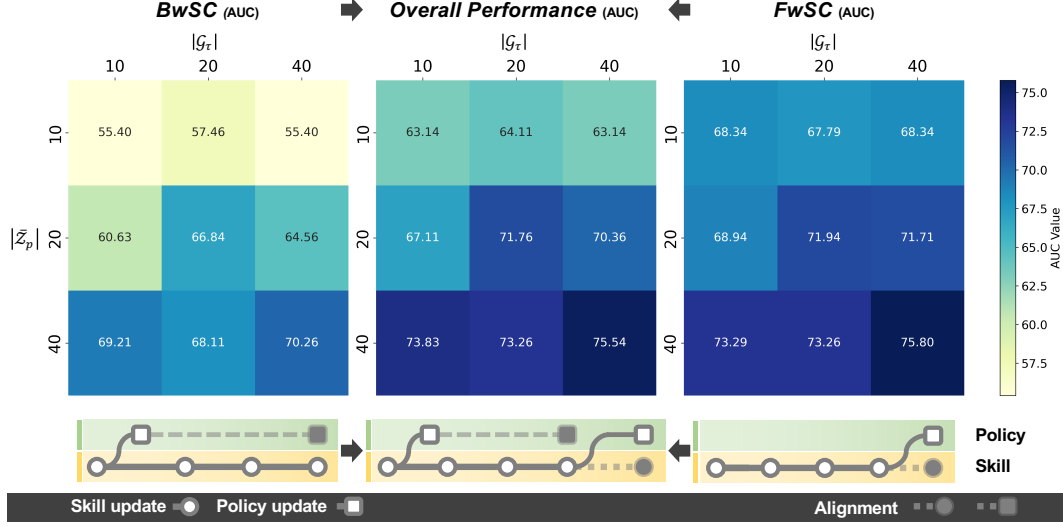


Figure 8: Analysis with varying  $|\mathcal{G}_\tau|$  and  $|\tilde{\mathcal{Z}}_p|$  in Franka Kitchen environment.  $|\mathcal{G}_\tau|$  denotes the number of subtasks into which expert demonstration data  $\mathcal{D}_\tau$  are segmented.  $|\tilde{\mathcal{Z}}_p|$  indicates the number of skills derived from the datastream  $\mathcal{D}_p$  at each phase  $p$ . (Left): Area Under the Curve (AUC) results for Backward Skill Compatibility (BwSC), which evaluates performance of existing policies with newly updated skills. (Right): AUC results for Forward Skill Compatibility (FwSC), assessing how effectively newly trained policies utilize previously learned skills. (Center): Overall performance combining both BwSC and FwSC scenarios.

743 within the BwSC evaluation group. This suggests that an insufficient number of subtask prototypes  
 744 may negatively impact initial policy performance in the BwSC setting of SIL.

### 745 C.3 Distance Analysis

Table 12: 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 Skill Incremental										
Baselines			Initial		BwSC (Initial)		Overall performance		FwSC (Synced)	
Type	Distance Type	Sub-clusters per prototype	FWT (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	BWT (%)	AUC (%)	FWT (%)
III	Mahalanobis	Fixed $K_{\tilde{\mathcal{Z}}}, K_{\tilde{\mathcal{G}}} = 4.00$ (100%)	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
		Auto $K_{\tilde{\mathcal{Z}}}, K_{\tilde{\mathcal{G}}} \approx 2.54$ (63.5%)	52.7 $\pm$ 3.0	16.6 $\pm$ 2.5	65.1 $\pm$ 2.0	19.6 $\pm$ 2.6	69.5 $\pm$ 1.6	22.7 $\pm$ 2.9	69.7 $\pm$ 1.4	80.6 $\pm$ 3.2
	Euclidean	Fixed $K_{\tilde{\mathcal{Z}}}, K_{\tilde{\mathcal{G}}} = 4$ (50%)	52.2 $\pm$ 4.7	5.5 $\pm$ 2.3	56.3 $\pm$ 3.1	14.3 $\pm$ 4.7	64.5 $\pm$ 0.8	23.2 $\pm$ 7.2	69.6 $\pm$ 0.7	81.0 $\pm$ 3.6
		Fixed $K_{\tilde{\mathcal{Z}}}, K_{\tilde{\mathcal{G}}} = 8$ (100%)	54.0 $\pm$ 2.6	5.6 $\pm$ 2.8	58.2 $\pm$ 1.8	14.4 $\pm$ 1.9	66.3 $\pm$ 1.7	23.1 $\pm$ 1.4	71.3 $\pm$ 2.0	82.9 $\pm$ 2.8

746 Table 12 reports the performance of SIL-C when using different distance metrics in its instance-based  
 747 classifier. In this experiment, all conditions were held constant except for the choice of distance metric  
 748 and the memory usage configuration, which was adjusted by varying the number of sub-clusters  
 749 per skill and subtask prototype. For Mahalanobis distance, we fixed the number of sub-clusters to  
 750 4 per prototype. Since each sub-cluster requires storing both  $\mu$  and  $\sigma$ , the memory cost amounted  
 751 to 8 vectors per prototype, which we treated as the 100% baseline. We also evaluated an automatic  
 752 clustering variant that determines the number of sub-clusters using the silhouette score. This approach  
 753 resulted in an average of 2.54 sub-clusters per prototype, corresponding to approximately 63.5% of

the baseline memory usage. In this configuration, the overall AUC was 3.2% lower than that of the fixed sub-cluster configuration.

For Euclidean distance, we also used 4 sub-clusters per prototype. To ensure comparable memory usage, only the mean  $\mu$  of each sub-cluster was stored. Distance thresholding was based on the 99th percentile of instance distances within each sub-cluster. Even when doubling the number of sub-clusters to 8, memory usage remained comparable to the Mahalanobis setting. Across configurations, the Euclidean distance variant consistently resulted in lower AUC performance in the overall evaluation, with a relative drop ranging from 7.66% to 10.17% compared to Mahalanobis distance. This gap is particularly pronounced in the *BwSC* setting, where the initial policy performance was significantly lower when using Euclidean distance (e.g., AUC of 56.3% vs. 66.8%). These results indicate that prototypes based on Euclidean distance are less effective at aligning the updated skill and subtask distributions with the trajectory distribution of the original policy.

These results suggest that Mahalanobis distance may offer a more effective mechanism for trajectory distribution matching in SIL-C, which is associated with improved skill-policy compatibility in the *BwSC* setting.

#### C.4 SIL-C Subtask Space Analysis

Table 13: 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 usage per task refers to the number of actually stored vectors (e.g.,  $\mu$  or  $\Sigma$ ), with relative usage at 100% corresponding to the configuration in Table 1. The left columns report initial forward transfer (*Initial*) and backward skill compatibility (*BwSC*), evaluating existing policies with updated skills. The right columns show forward skill compatibility (*FwSC*), assessing how effectively new policies utilize previously learned skills, and final forward transfer (*Final*). The central columns indicate overall performance by combining *BwSC* and *FwSC*.

Kitchen : Emergent Skill Incremental										
Ablations			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 $\pm$ 3.2	11.3 $\pm$ 2.4	55.9 $\pm$ 4.6	17.5 $\pm$ 1.2	62.4 $\pm$ 2.7	23.6 $\pm$ 4.2	65.1 $\pm$ 2.3	75.8 $\pm$ 4.4
	3	160 (100%)	44.4 $\pm$ 1.8	11.2 $\pm$ 2.5	52.8 $\pm$ 2.9	14.9 $\pm$ 2.2	57.2 $\pm$ 1.8	18.6 $\pm$ 3.3	58.4 $\pm$ 1.2	68.5 $\pm$ 4.0
	1	160 (100%)	43.1 $\pm$ 5.4	11.9 $\pm$ 2.8	52.0 $\pm$ 5.3	15.7 $\pm$ 3.1	56.5 $\pm$ 3.7	19.4 $\pm$ 4.8	57.6 $\pm$ 2.9	65.7 $\pm$ 3.6
Element-wise	5	1000 (625%)	50.3 $\pm$ 4.6	13.9 $\pm$ 4.3	60.7 $\pm$ 6.4	20.6 $\pm$ 1.6	66.6 $\pm$ 4.4	24.3 $\pm$ 5.3	68.5 $\pm$ 2.5	79.1 $\pm$ 3.4
	3	600 (375%)	48.9 $\pm$ 3.0	15.6 $\pm$ 4.6	60.6 $\pm$ 2.5	19.3 $\pm$ 4.2	65.4 $\pm$ 2.1	23.0 $\pm$ 3.9	66.1 $\pm$ 1.3	77.3 $\pm$ 3.6
	1	200 (125%)	42.3 $\pm$ 3.6	8.3 $\pm$ 3.3	48.5 $\pm$ 4.3	14.6 $\pm$ 2.4	54.8 $\pm$ 2.6	21.0 $\pm$ 3.8	58.0 $\pm$ 1.9	64.6 $\pm$ 3.2

Table 13 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.

#### C.5 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  $(\mu, \Sigma)$  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.

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

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

Table 14: 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		<i>kbls-Initial</i>		<i>btls-Initial</i>		<i>kbls-Final</i>		<i>btls-Final</i>	
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

795 Table 14 reports the inference time (mean and variance) of the hierarchical model, consisting of the  
796 policy, interface, and skill decoder, in the Kitchen *Emergent* Skill Incremental scenario. All timings  
797 were measured on an AMD Ryzen 9 7950X3D CPU with a single NVIDIA RTX 4090 GPU, running  
798 Ubuntu 22.04, CUDA 12.4, and driver version 550.144.03. The *Initial* phase ( $p = 1$ ) corresponds  
799 to the evaluation after 20 skill prototypes were generated, while the *Final* phase ( $p = 4$ ) reflects  
800 evaluation after the final phase with a total of 80 skills. Compared to Type I, the mean inference time  
801 of Type III increased by 0.953 ms and 1.094 ms in the *kbls-Initial* and *kbls-Final*, respectively. In the  
802 *btls-Initial*, the mean time decreased by 0.335 ms, while in the *btls-Final*, it increased by 0.486 ms.  
803 While a greater number of skills generally leads to longer retrieval time, the additional overhead was  
804 minimal compared to overall policy and skill decoder inference time, and variation due to runtime  
805 system state was more prominent in practice.