Reconfiguring Procedural Knowledge for Compositional Robot Skill Adaptation

Daehee Lee[♠] **Dongsu Lee**[◇] **Sanghyun Ahn**[♠*] **Minjong Yoo**[♣] **Honguk Woo**[♠†]
[♠]Sungkyunkwan University [◇]University of Texas at Austin [♣]Samsung Electronics
dulgi7245@skku.edu, hwoo@skku.edu

Abstract

General-purpose behavior models increasingly internalize broad procedural knowledge during training, making their adaptation to new goals and skill interfaces less a problem of learning entirely new behaviors than of reconfiguring existing competence. We study this capability as ***reconfigurability***, the ability to realize new behaviors by cost-efficiently restructuring existing components and relations under ***sparse governing information***, where task specifications provide high-level goals and procedure specifications expose a limited set of reusable factors, leaving the concrete bindings and compositions required for execution underdetermined. We propose `RecPo`, a framework that represents skills task-agnostically through multiple procedure views and materializes executable skill sequences by composing stored transitions to realize task-specific changes. RecPo maintains a procedure knowledge base of skill-induced transitions, with each view exposing a reusable factor and preserving local skill connectivity. At deployment, it interprets task specifications through the same views and resolves the resulting compositional queries over the stored relations to bind skills for execution. On long-horizon tasks in an extended Franka Kitchen benchmark, `RecPo` achieves 91.9% ordered completion with 1.0 ms of task inference per episode across varying procedure support and task-record sparsity. Real-robot studies further demonstrate adaptation through local memory edits without task-specific retraining.

1 Introduction

Reconfigurability is a core systems-level capability for lifelong embodied agents deployed in open-ended physical environments, enabling cost-effective adaptation under the constrained computational and physical resources available on-site [1–3]. Since design-time knowledge cannot cover every case [4] and runtime exposes only a partial and shifting subset of deployment conditions, agents must locally update existing competence as user specifications over behaviors and tasks change. These limitations compound for embodied agents, whose deployments take place in physical environments where spatial layouts, object properties, and physical dynamics vary across sites and shift over time.

Modern Vision-Language-Action (VLA) models [5–9] consolidate broad procedural knowledge at training time, leaving deployment to local adaptation. Performing such adaptation in practice, however, remains difficult: full retraining is costly and prone to catastrophic forgetting [10, 11], while ad hoc conditioning offers only transient adjustments [12]. Therefore, reconfiguration is not only about adaptation but also about externalizing this procedural knowledge into a form governable through its environment-grounded effects, to enable cost-effective updates at deployment [13–17].

Recent efforts toward reconfigurability increasingly rely on external memories of prior behaviors, in which adaptation involves adding, removing, or reweighting components rather than updating model

[*]Sanghyun Ahn is currently a visiting student at Carnegie Mellon University.
[†]Corresponding author: Honguk Woo (hwoo@skku.edu).

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

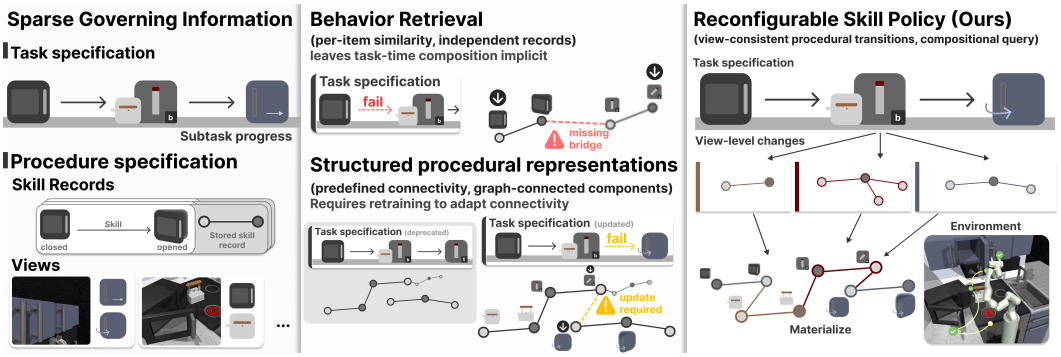


Figure 1: Conceptual overview of *reconfigurable skill policy* under *sparse governing information*. *(left)* Sparse task evidence specifies desired changes, while procedure-side evidence provides task-agnostic skill records observed through multiple procedure views. *(middle)* Behavior retrieval accesses stored records independently, whereas structured procedural approaches expose connectivity among reusable behaviors. *(right)* `RecPo` converts observed view-level changes into a compositional query and materializes a task-specific executable skill sequence over view-consistent procedural transitions.

parameters, giving rise to two complementary lines of research. Behavior retrieval organizes prior experiences as independently accessible records, either by retrieving trajectories [18–27] or by routing over learned skill embeddings and prototypes [28–33], allowing individual components to be added or removed without restructuring the entire memory. Structured procedural representations [34–37] instead encode explicit connectivity among reusable behavior components, supporting compositional access where a target instruction or goal is addressed by combining connected components.

However, these approaches face distinct challenges under *sparse governing information*, where task goals, reusable procedural factors, and prior experiences provide fragmented evidence that does not enumerate their possible task-specific combinations. Under these conditions, task demonstrations provide sparse snapshots of desired state changes, while procedure records capture skill-induced transitions realizable by the underlying behavior model without specifying how to compose them for execution. (i) Behavior retrieval typically matches stored experiences independently, leaving the transition compatibility between retrieved records under-specified. (ii) Structured procedural approaches explicitly represent transition connectivity, but often encode it within a single state or skill abstraction, which can obscure factor-specific transition relations. Reconfiguration of robot behavior therefore requires a structure that preserves factor-specific transition connectivity across varying contexts and supports task-conditioned composition of executable skill paths.

To address this gap, we propose `RecPo`, a multi-view update-and-compose framework for reconfigurable skill policies that maintains editable procedure and task memories to materialize task-specific executable skill sequences. In `RecPo`, the *Procedure Knowledge Base* organizes task-agnostic skill-induced transition records into view-local graphs, preserving skill connectivity and effect evidence across reusable factors of variation. At test time, *Task Materialization* uses these views to abstract the task demonstration into a compositional query of supported view-level changes, then resolves the query into an executable skill sequence by tracing compatible transitions across stored graphs.

We evaluate `RecPo` on an extended Franka Kitchen benchmark with held-out long-horizon tasks requiring compositions beyond stored skill records. By varying skill-context diversity, view resolution, and task-demonstration waypoint gaps, we assess reconfiguration under sparse procedure and task evidence. `RecPo` achieves 91.9% ordered completion at 1.0 ms of task inference per episode, outperforming retrieval and skill-chaining baselines. Real-robot studies further demonstrate deployment-time adaptation through local memory updates.

Our contributions are as follows:

- We formulate *reconfigurable skill policy* under *sparse governing information* as a cost-aware update-and-compose problem over editable procedure and task memories.
- We propose `RecPo`, a framework that composes task-agnostic skill records into task-specific executable skill sequences under a supplied procedure-view interface.
- We design a controlled protocol on an extended Franka Kitchen isolating procedure- and task-side sparseness, with a real-robot study of deployment-time adaptation.

2 Related Work

Memory-based policy adaptation in robot learning. Recent robot-learning systems increasingly treat memory as an external mechanism for cost-effective policy adaptation, rather than only as hidden state or model parameters. One line retrieves relevant demonstrations or experiences at deployment time and executes them through replay, alignment, or visual correspondence [38–40]. Another line uses retrieval to curate, fuse, or bootstrap task-relevant data from large-scale experience, improving few-shot generalization and robustness [18–27]. Recent robot policies further connect memory with in-context adaptation, retrieving task-relevant keyframes or sub-trajectories as context for VLA controllers and decision-transformer policies [14, 41, 42]. These in-context uses bring memory closer to policy execution, but still leave procedural recomposition inside the retrieved context or policy behavior rather than exposing it as a memory operation. This makes changes in object bindings, goal orders, contexts, or skill granularities difficult to update, inspect, and measure at test time.

Routing-based policy reconfiguration. In lifelong learning, such retrieval capabilities integrate external memory into learning and decision-making, separating generalized skill knowledge in pre-trained policies from task-specific composition delegated to test-time routing [11]. These approaches treat memory as support for selecting behaviors, routing over an expandable set of expert modules via similarity matching [28–31], merging retrieved skills into connected behaviors [33], or matching trajectory distributions at runtime to reconcile modules trained at different stages [32]. However, collapsing diverse factors of variation into a single embedding space coarsens the state abstraction, and routing it from sparse governing information limits forward transfer [43, 44]. This motivates decomposing behavior memory along partial views with local connectivity within each view.

Structured procedural representations. A complementary line exposes connectivity among reusable behaviors explicitly. Rooted in the options framework [45], these methods chain temporally extended primitives through their initiation and termination conditions to form explicit procedural structure [46–48]. A parallel direction composes options through learned models or weighted combinations of primitives [49–51]. Building on this connectivity view, recent works organize behavior memory as graphs over states, transitions, or skills and search them for temporal connectivity, including deep skill graphs over discovered options [48, 52], graph-based retrieval and stitching for goal-conditioned imitation [34, 35], and skill composition over learned operators [36, 37]. However, this connectivity is built along a single axis, typically temporal adjacency, and is fused with its underlying representation, leaving no handle to reorganize the same records under sparse task evidence. We propose `RecPo`, which stores skill-induced records under multiple partial views with local connectivity and materializes an executable structure across views when a task arrives.

3 Preliminaries

Controlled semi-Markov process [45]. We model the environment as a *controlled Markov process* (CMP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}_{\mathcal{A}})$ with state space $\mathcal{S}$, primitive action space $\mathcal{A}$, and one-step transition kernel $\mathcal{P}_{\mathcal{A}}(s' \mid s, a)$. Given a set of skills Ω, where each $\omega \in \Omega$ is a temporally extended controller over $\mathcal{A}$, the environment induces a *controlled semi-Markov process* (CSMP) $\mathcal{M}_{\Omega} = (\mathcal{S}, \Omega, \mathcal{P}_{\Omega})$, with semi-Markov transition kernel $\mathcal{P}_{\Omega}(s', k \mid s, \omega)$ over termination state $s' \in \mathcal{S}$ and duration $k \in \mathbb{N}_+$.

SMDP homomorphism [53–55]. For a given task τ, equipping the CSMP $\mathcal{M}_{\Omega}$ with a task-dependent reward $\mathcal{R}_{\tau} : \mathcal{S} \times \Omega \times \mathbb{N}_+ \to \mathbb{R}$, where $\mathcal{R}_{\tau}(s, \omega, k)$ denotes the reward of executing skill ω from state s for duration k, yields the task-specific SMDP $\mathcal{M}_{\Omega,\tau} = (\mathcal{S}, \Omega, \mathcal{P}_{\Omega}, \mathcal{R}_{\tau})$. A task-preserving abstraction of $\mathcal{M}_{\Omega,\tau}$ is specified by an SMDP homomorphism $h_{\tau} = (f_{\tau}, \{g_{\tau,s}\}_{s \in \mathcal{S}})$ from $\mathcal{M}_{\Omega,\tau}$ to an abstract SMDP $\bar{\mathcal{M}}_{\Omega,\tau}$, where $f_{\tau} : \mathcal{S} \to \bar{\mathcal{S}}_{\tau}$ is a surjective state map and $g_{\tau,s} : \Omega \to \bar{\Omega}_{\tau}$ is a state-dependent skill map. The map h_{τ} is an SMDP homomorphism if concrete state-skill pairs mapped to the same abstract state-skill pair induce identical aggregate semi-Markov transition kernels and task rewards. Specifically, for any $(s_1, \omega_1), (s_2, \omega_2) \in \mathcal{S} \times \Omega$ satisfying $f_{\tau}(s_1) = f_{\tau}(s_2)$ and $g_{\tau,s_1}(\omega_1) = g_{\tau,s_2}(\omega_2)$, the following condition holds for all $\bar{s}' \in \bar{\mathcal{S}}_{\tau}$ and $k \in \mathbb{N}_+$:

$$\sum_{s' : f_{\tau}(s') = \bar{s}'} \mathcal{P}_{\Omega}(s', k \mid s_1, \omega_1) = \sum_{s' : f_{\tau}(s') = \bar{s}'} \mathcal{P}_{\Omega}(s', k \mid s_2, \omega_2), \quad \mathcal{R}_{\tau}(s_1, \omega_1, k) = \mathcal{R}_{\tau}(s_2, \omega_2, k). \quad (1)$$

These conditions induce the abstract SMDP $\bar{\mathcal{M}}_{\Omega,\tau} = (\bar{\mathcal{S}}_{\tau}, \bar{\Omega}_{\tau}, \bar{\mathcal{P}}_{\tau}, \bar{\mathcal{R}}_{\tau})$.

4 Problem Formulation

We formalize a reconfigurable skill policy as an 'update-and-compose' rule over editable procedure and task memories, producing task-specific executable realizations via task-local abstractions.

Reconfigurable skill policy. Let $p \in \mathbb{N}_+$ index discrete phases, with superscript $(\cdot)^p$ indicating phase p. When no ambiguity arises, quantities without superscripts denote their current phase-p values. At each phase, two specifications $(\varphi_\Omega^p, \varphi_\tau^p)$ are given. The procedure-side specification φ_Ω^p supplies evidence and structural information about the currently realizable skills Ω, and the task-side specification φ_τ^p supplies task-state evidence and goal-side updates for the task τ under consideration [56]. The reconfigurable skill policy $\pi_{\Omega,\tau}^p$ maintains a paired procedure and task memory $(\mathcal{K}_\Omega^p, \mathcal{K}_\tau^p)$ and a composition map ϕ. The phase specifications update the memory pair, and the updated pair is composed as follows:

$$(\mathcal{K}_\Omega^p, \mathcal{K}_\tau^p) = \Big(U_\Omega(\mathcal{K}_\Omega^{p-1}, \varphi_\Omega^p), U_\mathcal{T}\big(\mathcal{K}_\tau^{p-1}, \varphi_\tau^p\big)\Big), \qquad \hat{h}_\tau^p = \phi(\mathcal{K}_\Omega^p, \mathcal{K}_\tau^p, \tau)\,. \tag{2}$$

Accordingly, the procedure update U_Ω instantiates and revises the currently realizable set of skill Ω and its induced phase-local CSMP $\mathcal{M}_\Omega^p$, while the task update $U_\mathcal{T}$ revises the task-side memory and the specification associated with τ. The composition map ϕ maps the updated memory pair and task τ to $\hat{h}_\tau^p$, which induces an abstract SMDP $\bar{\mathcal{M}}_{\Omega,\tau}^p$ by (1).

In practice, ϕ is realized as a composition of partial homomorphisms [54], so that $\hat{h}_\tau^p$ and $\bar{\mathcal{M}}_{\Omega,\tau}^p$ are approximations of an SMDP homomorphism and its induced abstract SMDP rather than exact ones. At execution time, an abstract decision $\bar{\pi}_\tau^p(\bar{\omega} \mid f_\tau^p(s))$ is grounded into a concrete skill through a task-conditioned realization rule that selects an executable $\omega \in \Omega$ consistent with the abstract skill $\bar{\omega}$ and the current state s, yielding the reconfigurable skill policy $\pi_{\Omega,\tau}^p(\omega \mid s)$.

Objective. Let $\mathcal{T}^p$ denote the set of tasks considered at phase p. The procedure-side specification φ_Ω^p is shared across $\mathcal{T}^p$, while each $\tau \in \mathcal{T}^p$ is associated with its own task-side specification φ_τ^p. Under changing environments and skill interfaces, the objective is to find update-and-compose operators $(U_\Omega^*, U_\mathcal{T}^*, \phi^*)$ that maximize per-task return while accounting for materialization and update costs:

$$(U_\Omega^*, U_\mathcal{T}^*, \phi^*) \in \underset{U_\Omega, U_\mathcal{T}, \phi}{\arg\max} \sum_p \left[\sum_{\tau \in \mathcal{T}^p} \Big(J_\tau(\pi_{\Omega,\tau}^p; \mathcal{M}_{\Omega,\tau}^p) - \mathcal{C}_\phi^p(\tau) \Big) - \mathcal{C}_U^p \right]. \tag{3}$$

Here, the return $J_\tau(\pi_{\Omega,\tau}^p; \mathcal{M}_{\Omega,\tau}^p)$ is evaluated on the phase-p task SMDP, with $\pi_{\Omega,\tau}^p$ defined by (2). The task-wise cost $\mathcal{C}_\phi^p(\tau)$ accounts for applying ϕ to materialize the task-specific abstraction, while the phase-wise cost $\mathcal{C}_U^p$ accounts for updating the procedure and task-side memories at phase p.

5 Methods

Following the update-and-compose formulation, `RecPo` instantiates (2) through two named memory interfaces that separate procedure storage from task-conditioned composition. Across these interfaces, multiple procedure views preserve distinct state equivalences over the same skill-induced transitions, allowing task materialization to combine task-relevant distinctions into a task-specific abstraction.

(i) Procedure knowledge base. The *Procedure Knowledge Base* realizes the procedure-memory pair $(\mathcal{K}_\Omega^p, U_\Omega)$. It updates and stores skill-induced transition evidence as procedure records, reads them through partial procedure views, and organizes the resulting view-projected records into skill-transition graphs and correlation-marked records. This yields a persistent procedure substrate whose reusable executable transitions support later reconfiguration.

(ii) Task materialization. The task-memory pair $(\mathcal{K}_\tau^p, U_\mathcal{T})$ maintains current task-state evidence. The composition map ϕ is realized by *Task Materialization*, which folds this evidence into the same canonical view-node space, forms task-query abstractions, and completes the induced query over the procedure substrate as current-state-conditioned skill paths. The resulting executable approximation of the task quotient carries prefix states and masks for grounding and local rematerialization during execution. Figure 2 provides an overview.

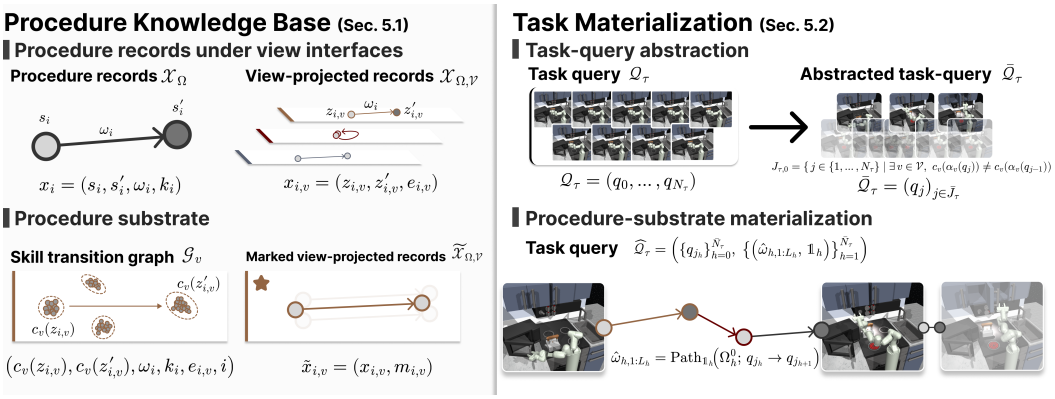


Figure 2: RecPo realizes update-and-compose through two memory interfaces. *(left)* The *Procedure Knowledge Base* turns procedure records and view interfaces into a persistent procedure substrate of skill-transition graphs and salience marks. *(right) Task Materialization* turns a task query into an abstracted query over the stored substrate and materializes it as an executable skill path.

5.1 Procedure Knowledge Base

Procedure records under view interfaces. At phase p, the procedure-side evidence φ_Ω^p provides observed executions under the current skills Ω. Let $\mathcal{I}_p$ with $|\mathcal{I}_p| = N_p$ denote the globally unique identifiers assigned to records newly added at phase p. RecPo updates the procedure-record set $\mathcal{X}_\Omega^p$ by appending a record x_i for each $i \in \mathcal{I}_p$:

$$\mathcal{X}_\Omega^p = \mathcal{X}_\Omega^{p-1} \cup \{x_i\}_{i \in \mathcal{I}_p}, \qquad x_i = (s_i, s'_i, \omega_i, k_i). \tag{4}$$

Here, $s_i, s'_i \in \mathcal{S}$ are segment endpoints, $\omega_i \in \Omega$ is the executed skill, and $k_i \in \mathbb{N}_+$ is its duration. Each record preserves executable transition evidence before it is organized for task-specific materialization.

Since the same record may be reused along different aspects of its transition under different tasks, RecPo reads each record through partial procedure views $\mathcal{V}$, with each view preserving one such aspect for task materialization. The view set $\mathcal{V}$ is supplied as part of the procedure-side specification, either specified directly or induced from state factorization, and each view $v \in \mathcal{V}$ provides a *view interface* consisting of a projection $\alpha_v : \mathcal{S} \to \mathcal{Z}_v$ and a view-local effect map δ_v on $\mathcal{Z}_v$. Pairing each record $x_i \in \mathcal{X}_\Omega$ with each $v \in \mathcal{V}$ yields the view-projected record collection $\mathcal{X}_{\Omega,\mathcal{V}}$,

$$\mathcal{X}_{\Omega,\mathcal{V}} = \{x_{i,v} \mid x_i \in \mathcal{X}_\Omega,\ v \in \mathcal{V}\}, \qquad x_{i,v} = (z_{i,v}, z'_{i,v}, e_{i,v}). \tag{5}$$

Here, $x_{i,v}$ is the view-projected record obtained by reading x_i through view v, with projected start $z_{i,v} = \alpha_v(s_i)$, projected endpoint $z'_{i,v} = \alpha_v(s'_i)$, and view-local effect $e_{i,v} = \delta_v(z_{i,v}, z'_{i,v})$.

Procedure substrate. To expose local connectivity among projected procedure effects, RecPo canonicalizes the view-projected records into per-view skill-transition graphs. For each view $v \in \mathcal{V}$, data-adaptive clustering of projected record endpoints (*e.g.*, DBSCAN [57]) induces canonical view nodes $\mathcal{N}_v$ and a canonicalization map $c_v : \mathcal{Z}_v \to \mathcal{N}_v$. Each view-projected record $x_{i,v}$ then contributes a directed edge between its canonicalized endpoints, forming the view-level skill-transition graph $\mathcal{G}_v$:

$$\mathcal{G}_v = (\mathcal{N}_v, \mathcal{E}_v), \qquad \mathcal{E}_v = \left\{\left(c_v(z_{i,v}), c_v(z'_{i,v}), \omega_i, k_i, e_{i,v}, i\right) \mid x_i \in \mathcal{X}_\Omega\right\}. \tag{6}$$

Here, each edge in $\mathcal{E}_v$ runs from the canonicalized start $c_v(z_{i,v})$ to the canonicalized endpoint $c_v(z'_{i,v})$, carrying the executed skill ω_i, duration k_i, view-local effect $e_{i,v}$, and procedure-record pointer i. The collection $\mathcal{G}_{\Omega,\mathcal{V}} = \{\mathcal{G}_v\}_{v \in \mathcal{V}}$ forms the view-level skill-transition graph collection.

The same view-projected records are also annotated with salience evidence. For each $x_{i,v}$, RecPo computes a salience mark $m_{i,v} = \Gamma(x_{i,v}; \mathcal{X}_{\Omega,\mathcal{V}})$, which measures local skill-label consistency among neighboring view-local effects. This induces the marked view-projected records collection $\bar{\mathcal{X}}_{\Omega,\mathcal{V}}$:

$$\bar{\mathcal{X}}_{\Omega,\mathcal{V}} = \{\bar{x}_{i,v} \mid x_{i,v} \in \mathcal{X}_{\Omega,\mathcal{V}}\}, \qquad \bar{x}_{i,v} = (x_{i,v}, m_{i,v}). \tag{7}$$

Together, the pair $(\mathcal{G}_{\Omega,\mathcal{V}},\ \bar{\mathcal{X}}_{\Omega,\mathcal{V}})$ defines the *procedure substrate* stored in $\mathcal{K}_\Omega$.

5.2 Task Materialization

To realize $\pi^p_{\Omega,\tau}$, RecPo instantiates the composition map ϕ in two stages over the task query $\mathcal{Q}_\tau = (q_0, \ldots, q_{N_\tau})$ from $\mathcal{K}_\tau$, a sequence of task-side waypoints. The first stage abstracts $\mathcal{Q}_\tau$ into a query against the precomputed procedure substrate $(\mathcal{G}_{\Omega,\mathcal{V}}, \bar{\mathcal{X}}_{\Omega,\mathcal{V}})$, and the second materializes the abstracted query as executable skill bindings.

Task-query abstraction. To form the initial query $\mathcal{Q}_{\tau,0}$, RecPo projects each q_j into its per-view node assignment $\{c_v(\alpha_v(q_j))\}_{v\in\mathcal{V}}$ and collects *decision points*, the steps at which at least one view node differs from the preceding step, into $\mathcal{Q}_{\tau,0}$,

$$\mathcal{Q}_{\tau,0} = (q_j)_{j\in J_{\tau,0}}, \qquad J_{\tau,0} = \{\, j \in \{1,\ldots,N_\tau\} \mid \exists\, v \in \mathcal{V},\ c_v(\alpha_v(q_j)) \neq c_v(\alpha_v(q_{j-1}))\,\}. \quad (8)$$

Here, $\mathcal{Q}_{\tau,0}$ is the subsequence of $\mathcal{Q}_\tau$ at the decision points indexed by $J_{\tau,0}$.

To refine the initial query $\mathcal{Q}_{\tau,0}$ into a substrate-supported subsequence, RecPo traverses $\mathcal{Q}_\tau$ step by step. From the current decision point, $\mathcal{G}_{\Omega,\mathcal{V}}$ exposes which view-node transitions are reachable through stored records, while $\bar{\mathcal{X}}_{\Omega,\mathcal{V}}$ supplies the marked transition evidence used to score whether the observed task displacement agrees with stored view-local effects. The next abstracted decision point is selected from candidate steps containing a changed view in which some stored skill attains its highest mean salience mark $m_{i,v}$ over its records. Iterating along $\mathcal{Q}_\tau$ and retaining the decision points that stored records support as segment boundaries yields the abstracted query $\bar{\mathcal{Q}}_\tau$,

$$\bar{\mathcal{Q}}_\tau = (q_j)_{j\in\bar{J}_\tau}, \qquad \bar{J}_\tau = \{j_0, j_1, \ldots, j_{\bar{N}_\tau}\} \subseteq \{0\} \cup J_{\tau,0}, \qquad 0 = j_0 < \cdots < j_{\bar{N}_\tau} \leq N_\tau. \quad (9)$$

Here, $\bar{\mathcal{Q}}_\tau$ collects the decision points of $\mathcal{Q}_\tau$ supported by the procedure substrate, indexed by $\bar{J}_\tau$ with start index 0 prepended, partitioned into $\bar{N}_\tau$ segments for executable binding.

Procedure-substrate materialization. Each consecutive pair $(q_{j_h}, q_{j_{h+1}})$ in $\bar{\mathcal{Q}}_\tau$ specifies a local segment with active views $\mathcal{V}_h \subseteq \mathcal{V}$, the views in which the canonical node changes, and a desired view-local effect on each record, with the indicator $\mathbb{1}_h = \mathbb{1}[\mathcal{V}_h]$ used at execution time for grounding. Restricting the marked records $\bar{\mathcal{X}}_{\Omega,\mathcal{V}}$ to $\mathbb{1}_h$ yields the admissible skill set $\Omega_h \subseteq \Omega$ with stored view-local effects on the active views of the segment.

For each skill in Ω_h, RecPo selects the view in which the records of the skill attain the highest mean salience mark. In this view, the stored view-local effect most reliably distinguishes the skill from other stored skills. RecPo keeps a skill when, in this view, its typical stored effect points in the direction of the change that the segment demands. Because each skill is compared only in its own view, skills that account for different parts of this change are kept together in the candidate set $\Omega^0_h \subseteq \Omega_h$.

RecPo then binds each segment by applying the path operator to the candidate skill set under $\mathbb{1}_h$,

$$\hat{\omega}_{h,1:L_h} = \mathrm{Path}_{\mathbb{1}_h}\big(\Omega^0_h;\, q_{j_h} \to q_{j_{h+1}}\big), \qquad L_h = |\Omega^0_h|. \quad (10)$$

Here, $\mathrm{Path}_{\mathbb{1}_h}$ returns a chained executable skill sequence over Ω^0_h between the segment endpoints, ordering all selected skills so that each starts near where the previous one ends on the active views. The candidate selection rule and path-search objective are specified in Appendix A.3.

Finally, the segment bindings form the materialized object

$$\widehat{\mathcal{Q}}_\tau = \Big(\{q_{j_h}\}_{h=0}^{\bar{N}_\tau},\ \big\{\big(\hat{\omega}_{h,1:L_h},\, \mathbb{1}_h\big)\big\}_{h=0}^{\bar{N}_\tau-1}\Big) \quad (11)$$

is an executable approximation of the task quotient. The decision points $\{q_{j_h}\}$ carry the abstract decisions, and each segment h supplies an executable skill sequence $\hat{\omega}_{h,1:L_h}$ together with its active-view indicator $\mathbb{1}_h$ used for grounding. At execution, the runtime advances to segment $h{+}1$ only when the view-node transition under $\mathbb{1}_h$ is observed, and a local inconsistency triggers rematerialization of the affected segment alone.

6 Experiments

Our experiments evaluate whether RecPo can reconfigure policies under *sparse governing information*, manifesting in procedure specifications that under-determine how to compose stored skills, and in task specifications that under-determine which composition to produce. We additionally examine whether these gains transfer to a real robot. We structure the evaluation around three questions:

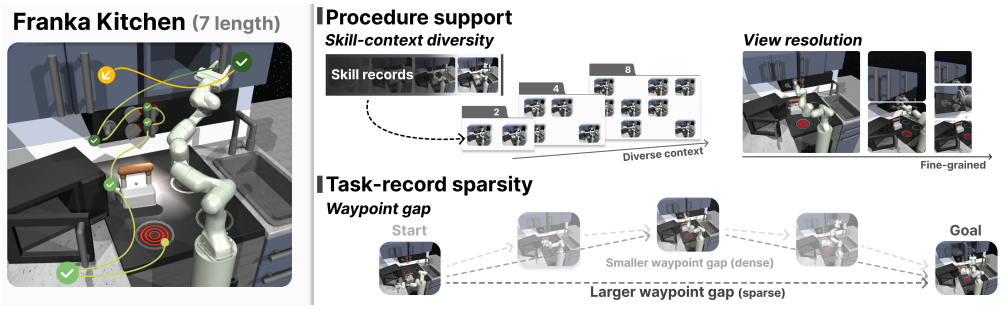


Figure 3: *(left)* Franka Kitchen with an example length-7 target task, held out from the procedure knowledge base. *(right)* Evaluation axes. *Procedure support* varies *skill-context diversity*, the number of length-4 task realizations populating the procedure knowledge base, and *view resolution*, the granularity of view projections supplied at inference. *Task-record sparsity* varies the *waypoint gap* H, the stride in subtask indices between consecutive task-record waypoints.

RQ1. Is `RecPo` robust across varying *skill-context diversity* and *view resolution*?

RQ2. Can `RecPo` recover executable composition under sparse task specifications?

RQ3. Can `RecPo` adapt a real-world robot through local procedure or task updates?

6.1 Experiment Settings

Environment. We build on Franka Kitchen [58, 59] by extending it with a broader set of causally independent [60] subtasks that admit varied executions, expanding the space of combinatorial long-horizon compositions and the diversity of contexts in which each skill is observed. We define realization length as the number of subtasks executed in sequence, and extend the original 24 length-4 realizations of Franka Kitchen up to length-7.

Evaluation protocol. We design the evaluation to stress-test the update-and-compose setting under sparse governing information by varying *procedure support* and *task-record sparsity*. The evaluation split uses length-4 realizations to construct the procedure knowledge base and introduces a held-out length-7 target task at test time whose complete skill composition is absent from the stored records.

Procedure support stresses the *procedure knowledge base* by sweeping *skill-context diversity*, the variety of contexts in which each skill is observed, over $\{4, 8, 16, 32\}$ length-4 subtask orderings populating the procedure knowledge base. For the view axis, the procedure-view interface is built from Kitchen-defined object-factor views and a robot-state view, with the finest setting separating them into up to eight atomic views. *View resolution* varies the number of available views over $\{3, 4, 5, 8\}$ by randomly merging object factors into composite views. *Task-record sparsity* stresses *Task Materialization* by sweeping the *waypoint gap* $H \in \{1, ..., 7\}$, the stride between consecutive task-record waypoints in subtask indices, with the final goal appended whenever the stride does not reach it. For H=4, this yields the waypoint list `[0,4,7]`. For each experimental configuration, we use 10 random seeds and 100 evaluation episodes per seed. We report *ordered completion* and per-episode wall-clock task inference time. Figure 3 provides an overview, with details in Appendix B.4.

Baselines. We evaluate all baselines under a shared reconfiguration setting that varies *procedure support* and *task-record sparsity* to assess how different composition mechanisms operate under *sparse governing information*. `Skill Retrieval` (`SR`) first computes a multi-view score for each candidate skill by matching each waypoint transition to the closest stored record for that skill in each procedure view, then independently reranks skills by the averaged view-wise score. `Skill Sequence Retrieval` (`SSR`) extends this retrieval to the combinatorial space of per-step skill candidates, ranking candidate sequences by aggregated retrieval scores without explicitly modeling inter-skill connectivity. `Skill Chaining` (`SC`) [46, 48] represents procedural connectivity through a skill-level transition graph. Each skill is summarized by canonical initiation and termination states estimated from stored endpoints, with graph edges weighted by termination-to-initiation distances.

Metrics. We report *ordered completion*, the fraction of environment-defined subtasks completed in the demonstrated order [61], and *task inference time*, the total time spent inferring an executable skill sequence per episode.

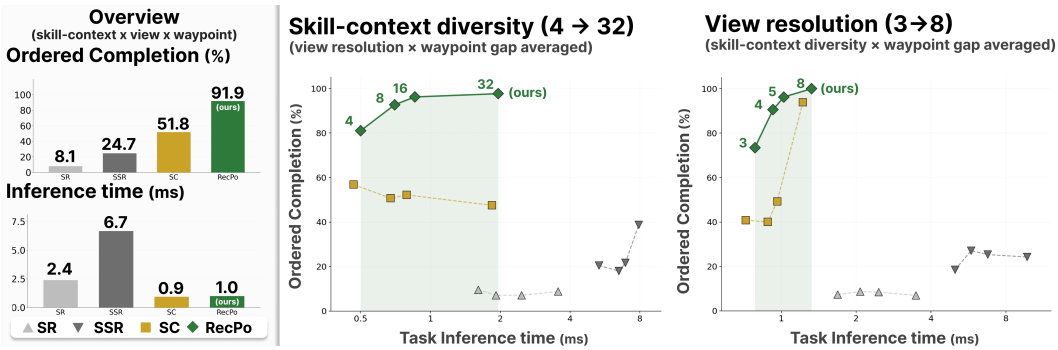


Figure 4: *Ordered completion*–task inference time trade-off under varying *procedure support* and *task-record sparsity*. *(left)* Overall comparison of the three baselines and RecPo, averaged across all evaluation conditions. *(right)* Trade-offs under varying *procedure support*, sweeping *skill-context diversity* and *view resolution* separately while averaging over the remaining axes.

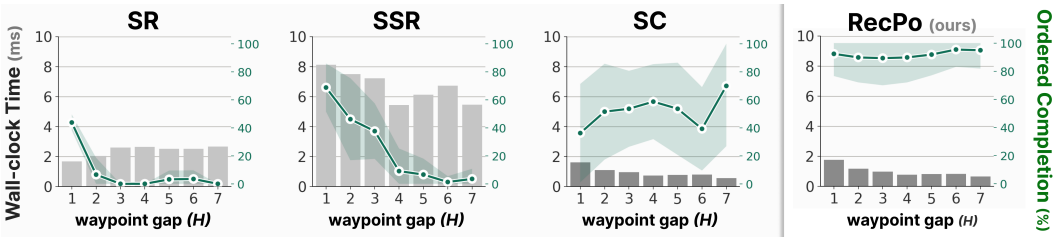


Figure 5: *Ordered completion* and task inference time under varying *task-record sparsity*. Results for the three baselines and RecPo are reported across *waypoint gap* H, averaged over *skill-context diversity* and *view resolution*. Gray bars indicate per-episode task inference time (ms, left axis), while green lines indicate *ordered completion* (%, right axis). Shaded bands indicate one standard deviation of *ordered completion* across procedure-support configurations at each H.

6.2 Procedure Knowledge Base

To answer **RQ1**, we evaluate RecPo and all baselines under sparse procedure specifications by varying *skill-context diversity* and *view resolution*, with smaller values of either instantiating sparser regimes. Figure 4 *(left)* shows that RecPo, which narrows candidates to view-consistent transitions, achieves the highest *ordered completion* at 91.9% with only 1.0 ms of task inference per episode. SSR widens the search to the combinatorial space of skill sequences but reaches 24.7% at 6.7 ms, suggesting that naive scaling of the search space offers limited gains without structural constraints on composition.

Figure 4 *(right)* shows that task inference time increases for every method with higher *skill-context diversity* or *view resolution*, while every RecPo configuration remains on the Pareto front of completion and time. SC declines from 56.9% to 47.5% as skill-context diversity increases and approaches RecPo only at the highest view resolution, since the skill-level graph of SC aliases endpoints across contexts, and higher view resolution still leaves hard negatives whose endpoints match but whose effects do not. RecPo instead structures salience marks with skill-transition graphs, so records remain distinguishable by the views they govern, keeping composition robust under sparse *procedure support*.

6.3 Task Materialization

To answer **RQ2**, we evaluate RecPo and all baselines under sparse task specifications by varying the *waypoint gap* $H \in \{1, \ldots, 7\}$ over the length-7 target task. Figure 5 shows that RecPo maintains 89.3–95.4% *ordered completion* across all H while its task inference time decreases from 1.8 to 0.7 ms. SR and SSR collapse as H increases, with SSR falling from 68.7% to at most 9.1% for $H \geq 4$, since similarity alone surfaces stored segments that do not realize demanded subtasks absent from the procedure knowledge base. Unlike retrieval, SC and RecPo do not degrade with sparser waypoints, since each waypoint fixes a segment to bind, and denser waypoints multiply the bindings over which errors accumulate. SC accumulates the aliasing identified in RQ1 across these bindings and ranges from 36.2% to 69.9%, whereas RecPo binds each segment to view-consistent transitions and keeps the accumulated error small.

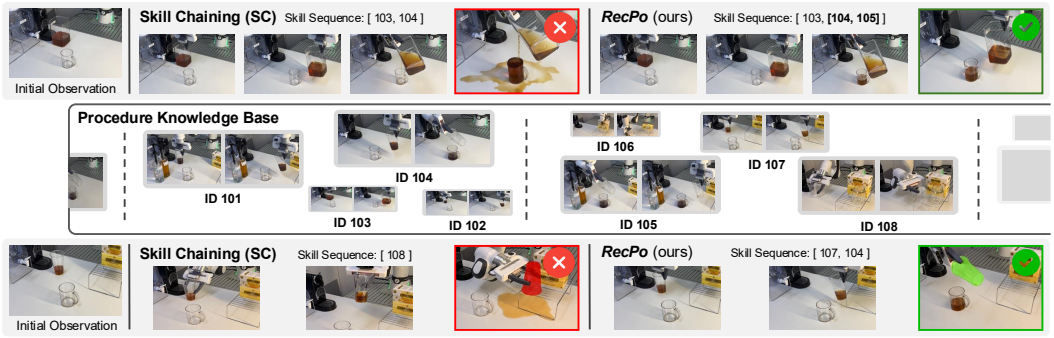


Figure 6: Real-world use cases with the shared goal *pour coffee*. Records 101 to 104 alone leave both cases unsolved, so the procedure knowledge base is locally updated with records 105 to 108 and a view for the target fill level, over which SC and RecPo compose. Bracketed record IDs denote one segment materialized as a multi-skill path. *(top)* The source holds more coffee than the target cup can take, so pouring must stop before the cup fills. *(bottom)* The scene shares its rack and box background with records 106 to 108, while pouring record 104 was recorded on a plain table.

6.4 Use Cases

To answer **RQ3**, we test on a real Franka robot whether local updates adapt RecPo along two axes without retraining, where view updates change what composition distinguishes and record updates change what it can bind, in the two cases of Figure 6.

In the first case of Figure 6 *(top)*, SC chains pouring record 104 alone and overflows the cup, reflecting the aliasing observed in RQ1. Even with the fill-level view added, its endpoint match still weighs the other views without salience, so stopping record 105, recorded in a different background, does not enter the chain. RecPo weighs record 105 by its salience on the fill-level view and rematerializes the pouring segment as a skill path chaining records 104 and 105.

In the second case of Figure 6 *(bottom)*, SC binds record 108, a hard negative that also pours but inverts the cup, since its endpoints match the shared background while its effect does not realize the demanded fill-level transition, and spills the contents. RecPo binds pouring record 104 by its effect on the fill-level view after record 107, so the background difference does not mislead the binding.

7 Conclusion

We present RecPo, which formulates the reconfiguration of skill policies as test-time composition over an editable and inspectable procedure-memory interface. RecPo stores skill-induced transition evidence as task-agnostic records and organizes them through procedure views. From sparse task evidence, RecPo materializes task-specific executable skill sequences without retraining. Results show that this structured materialization improves the success-cost trade-off of reconfiguration across sparse procedure support and sparse task specifications, supporting adaptation through recomposition.

Limitations and future directions. Our work investigates the reconfiguration of procedural knowledge through task-conditioned composition over supplied procedure views. This formulation assumes the availability of views that expose task-relevant skill effects and their connectivity. Inadequate view decomposition may obscure these relations and limit reconfigurability. These views can still be separated counterfactually to match the intent of the user without retraining, and extending this interface beyond object-centric factors and context generalization remains open. Beyond the views, each skill record keeps only its start and end states, which suffices for composing skills whose effects transfer across contexts but not for skills whose outcome depends on how they are executed, and editable records that also carry such execution conditions are a natural extension.

As a future direction, the editability and inspectability of the memory interface can be leveraged for reconfiguration across diverse scenarios, such as interactive updates on resource-limited embedded devices and guardrails, since composition realizes only transitions supported by the memory and editing its views and records therefore bounds what a robot can do.

Acknowledgment

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2025-02218768, Accelerated Insight Reasoning via Continual Learning (50%); RS-2026-25505492, High-Performance Research AI Computing Infrastructure Support at the 2 PFLOPS Scale (10%); RS-2025-25442569, AI Star Fellowship Support Program (Sungkyunkwan Univ.) (10%)), IITP-ICT Creative Consilience Program grant funded by the Korea government (MSIT) (IITP-2026-RS-2020-II201821) (10%), by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2026-25474409) (10%), and by Samsung Electronics Co., Ltd. (10%).

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 36 (NeurIPS)*, 2023.

[2] Young D Kwon, Jagmohan Chauhan, Hong Jia, Stylianos I Venieris, and Cecilia Mascolo. Lifelearner: Hardware-aware meta continual learning system for embedded computing platforms. In *Proceedings of the 21st ACM Conference on Embedded Networked Sensor Systems*, pages 138–151, 2023.

[3] Shahram Najam Syed, Yatharth Ahuja, Arthur Jakobsson, and Jeff Ichnowski. Expres-vla: Specializing vision-language-action models through experience replay and retrieval. *arXiv preprint arXiv:2511.06202*, 2025.

[4] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems. *Aaai/iaai*, 1999.

[5] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2025.

[6] Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, brian ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In Joseph Lim, Shuran Song, and Hae-Won Park, editors, *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 17–40. PMLR, 27–30 Sep 2025.

[7] NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots, 2025.

[8] Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, Danny Driess, Michael Equi, Adnan Esmail, Yunhao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev, Lachy Groom, Hunter Hancock, Karol Hausman, Gashon Hussein, Brian Ichter, Szymon Jakubczak, Rowan Jen, Tim Jones, Ben Katz, Liyiming Ke, Chandra Kuchi, Marinda Lamb, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Yao Lu, Vishnu Mano, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Charvi Sharma, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Will Stoeckle, Alex Swerdlow, James Tanner, Marcel Torne, Quan Vuong, Anna Walling, Haohuan Wang, Blake Williams, Sukwon Yoo, Lili Yu, Ury Zhilinsky, and Zhiyuan Zhou. $\pi^*_{0.6}$: a vla that learns from experience, 2025.

[9] Physical Intelligence, Bo Ai, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Greg Balke, Kevin Black, George Bokinsky, Shihao Cao, Thomas Charbonnier, Vedant Choudhary, et al. $\pi_{0.7}$: a steerable generalist robotic foundation model with emergent capabilities, 2026.

[10] Asher J. Hancock, Xindi Wu, Lihan Zha, Olga Russakovsky, and Anirudha Majumdar. Actions as language: Fine-tuning VLMs into VLAs without catastrophic forgetting. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026.

[11] Zheda Mai, Zanming Huang, Sooyoung Jeon, Jinsu Yoo, Colin Lee, Gengyu Zhang, Haoxuan Wang, Yifan Li, Yu Kong, Yan Yan, et al. A survey of continual learning for robotics in the foundation model era. 2026.

[12] Shuo Liu, Ishneet Sukhvinder Singh, Yiqing Xu, Jiafei Duan, and Ranjay Krishna. VLS: Steering pretrained robot policies via vision-language models. *arXiv preprint arXiv:2602.03973*, 2026.

[13] Jacky Liang, Mohit Sharma, Alex LaGrassa, Shivam Vats, Saumya Saxena, and Oliver Kroemer. Search-based task planning with learned skill effect models for lifelong robotic manipulation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6351–6357. IEEE, 2022.

[14] Thomas Schmied, Fabian Paischer, Vihang Prakash Patil, Markus Hofmarcher, Razvan Pascanu, and Sepp Hochreiter. Retrieval-augmented decision transformer: External memory for in-context RL, 2025.

[15] Xue Qin, Simin Luan, John See, Cong Yang, and Zhijun Li. Harnessing embodied agents: Runtime governance for policy-constrained execution. *arXiv preprint arXiv:2604.07833*, 2026.

[16] Tianle Xia, Lingxiang Hu, Yiding Sun, Ming Xu, Lan Xu, Siying Wang, Wei Xu, and Jie Jiang. Grasp: Graph-structured skill compositions for llm agents. *arXiv preprint arXiv:2604.17870*, 2026.

[17] Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, et al. Externalization in llm agents: A unified review of memory, skills, protocols and harness engineering. *arXiv preprint arXiv:2604.08224*, 2026.

[18] 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.

[19] Soroush Nasiriany, Tian Gao, Ajay Mandlekar, and Yuke Zhu. Learning and retrieval from prior data for skill-based imitation learning. In *Proceedings of The 6th Conference on Robot Learning*, 2023.

[20] Jyothish Pari, Nur Muhammad, Sridhar Pandian Arunachalam, and Lerrel Pinto. The surprising effectiveness of representation learning for visual imitation. *arXiv preprint arXiv:2112.01511*, 2021.

[21] 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.

[22] 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.

[23] Lihan Zha, Yuchen Cui, Li-Heng Lin, Minae Kwon, Montserrat Gonzalez Arenas, Andy Zeng, Fei Xia, and Dorsa Sadigh. Distilling and retrieving generalizable knowledge for robot manipulation via language corrections. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[24] 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.

[25] Sateesh Kumar, Shivin Dass, Georgios Pavlakos, and Roberto Martín-Martín. COLLAGE: Adaptive fusion-based retrieval for augmented policy learning. In *9th Annual Conference on Robot Learning*, 2025.

[26] Shivin Dass, Alaa Khaddaj, Logan Engstrom, Aleksander Madry, Andrew Ilyas, and Roberto Martín-Martín. DataMIL: Selecting data for robot imitation learning with datamodels. In *The Fourteenth International Conference on Learning Representations*, 2026.

[27] William Liang, Sam Wang, Hung-Ju Wang, Osbert Bastani, Yecheng Jason Ma, and Dinesh Jayaraman. Autonomous functional play with correspondence-driven trajectory warping. In *The Fourteenth International Conference on Learning Representations*, 2026.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] Daehee Lee, Dongsu Lee, TaeYoon Kwack, Wonje Choi, and Honguk Woo. Policy compatible skill incremental learning via lazy learning interface. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[33] Yuheng Lei, Sitong Mao, Shunbo Zhou, Hongyuan Zhang, Xuelong Li, and Ping Luo. Dynamic mixture of progressive parameter-efficient expert library for lifelong robot learning. *arXiv preprint arXiv:2506.05985*, 2025.

[34] Zhao-Heng Yin and Pieter Abbeel. Offline imitation learning through graph search and retrieval. *Robotics: Science and Systems*, 2024.

[35] Shuze Wang, Yunpeng Mei, Hongjie Cao, Yetian Yuan, Gang Wang, Jian Sun, and Jie Chen. Robust offline imitation learning through state-level trajectory stitching. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2025.

[36] Zixuan Chen, Junhui Yin, Yangtao Chen, Jing Huo, Pinzhuo Tian, Jieqi Shi, Yiwen Hou, Yinchuan Li, and Yang Gao. Deco: Task decomposition and skill composition for zero-shot generalization in long-horizon 3d manipulation. *IEEE Robotics and Automation Letters*, 2026.

[37] Yifei Simon Shao, Yuchen Zheng, Sunan Sun, Pratik Chaudhari, Vijay Kumar, and Nadia Figueroa. SymSkill: Symbol and skill co-invention for data-efficient and reactive long-horizon manipulation. *arXiv preprint arXiv:2510.01661*, 2025.

[38] Owen Kwon, Abraham George, Alison Bartsch, and Amir Barati Farimani. Rt-cache: Training-free retrieval for real-time manipulation. In *2025 IEEE-RAS 24th International Conference on Humanoid Robots (Humanoids)*, pages 1–8. IEEE, 2025.

[39] Norman Di Palo and Edward Johns. On the effectiveness of retrieval, alignment, and replay in manipulation. *IEEE Robotics and Automation Letters*, 9(3):2032–2039, 2024.

[40] Norman Di Palo and Edward Johns. Dinobot: Robot manipulation via retrieval and alignment with vision foundation models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2798–2805. IEEE, 2024.

[41] Jonathan Lee, Annie Xie, Aldo Pacchiano, Yash Chandak, Chelsea Finn, Ofir Nachum, and Emma Brunskill. In-context decision-making from supervised pretraining. In *ICML Workshop on New Frontiers in Learning, Control, and Dynamical Systems*, 2023.

[42] Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. Memer: Scaling up memory for robot control via experience retrieval. *arXiv preprint arXiv:2510.20328*, 2025.

[43] David Abel, Dilip Arumugam, Lucas Lehnert, and Michael Littman. State abstractions for lifelong reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 10–19. PMLR, 10–15 Jul 2018.

[44] Tesca Fitzgerald, Ashok Goel, and Andrea Thomaz. Abstraction in data-sparse task transfer. *Artificial Intelligence*, 300:103551, 2021.

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

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

[47] Akhil Bagaria and George Konidaris. Option discovery using deep skill chaining. In *International Conference on Learning Representations*, 2020.

[48] Akhil Bagaria, Jason K Senthil, and George Konidaris. Skill discovery for exploration and planning using deep skill graphs. In *International conference on machine learning*, pages 521–531. PMLR, 2021.

[49] David Silver and Kamil Ciosek. Compositional planning using optimal option models. In *Proceedings of the 29th International Conference on International Conference on Machine Learning*, 2012.

[50] André Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Shibl Mourad, David Silver, Doina Precup, et al. The option keyboard: Combining skills in reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[51] Lucas N. Alegre, Ana L. C. Bazzan, Andre Barreto, and Bruno Castro da Silva. Constructing an optimal behavior basis for the option keyboard. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[52] Akhil Bagaria, Anita De Mello Koch, Rafael Rodriguez-Sanchez, Sam Lobel, and George Konidaris. Intrinsically motivated discovery of temporally abstract graph-based models of the world. In *Reinforcement Learning Conference*, 2025.

[53] Balaraman Ravindran. Smdp homomorphisms: An algebraic approach to abstraction in semi markov decision processes. 2003.

[54] Balaraman Ravindran. *An algebraic approach to abstraction in reinforcement learning*. University of Massachusetts Amherst, 2004.

[55] Sahand Rezaei-Shoshtari, Rosie Zhao, Prakash Panangaden, David Meger, and Doina Precup. Continuous mdp homomorphisms and homomorphic policy gradient. *Advances in Neural Information Processing Systems*, 35:20189–20204, 2022.

[56] Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. A composable specification language for reinforcement learning tasks. *Advances in Neural Information Processing Systems*, 32, 2019.

[57] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996.

[58] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Proceedings of the Conference on Robot Learning*, 2019.

[59] 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.

[60] Núria Armengol Urpí, Marco Bagatella, Marin Vlastelica, and Georg Martius. Causal action influence aware counterfactual data augmentation. In *Proceedings of the 41st International Conference on Machine Learning*.

[61] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022.

[62] Silviu Pitis, Elliot Creager, and Animesh Garg. Counterfactual data augmentation using locally factored dynamics. In *Advances in Neural Information Processing Systems*, volume 33, pages 3976–3990, 2020.

[63] Caleb Chuck, Fan Feng, Carl Qi, Chang Shi, Siddhant Agarwal, Amy Zhang, and Scott Niekum. Null counterfactual factor interactions for goal-conditioned reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025.

[64] Oriane Siméoni, Huy V. Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, Francisco Massa, Daniel Haziza, Luca Wehrstedt, Jianyuan Wang, Timothée Darcet, Théo Moutakanni, Leonel Sentana, Claire Roberts, Andrea Vedaldi, Jamie Tolan, John Brandt, Camille Couprie, Julien Mairal, Hervé Jégou, Patrick Labatut, and Piotr Bojanowski. DINOv3, 2025.

[65] Evgenii Opryshko, Junwei Quan, Claas Voelcker, Yilun Du, and Igor Gilitschenski. Test-time graph search for goal-conditioned reinforcement learning. *arXiv preprint arXiv:2510.07257*, 2025.

[66] Seungho Baek, Taegeon Park, Jongchan Park, Seungjun Oh, and Yusung Kim. Graph-assisted stitching for offline hierarchical reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 2391–2408, 2025.

Appendix

Contents

A `RecPo` **15**

A.1 Procedure Views 15

A.2 Procedure Knowledge Base 15

A.3 Task Materialization 16

B Experiment Details **17**

B.1 Environment 17

B.2 Baseline Implementations 19

B.3 Computing Details 20

B.4 Evaluation Protocol and Aggregation 20

B.5 Main Evaluation Results 21

B.6 Real-World Evaluation 24

C Additional Experiments **25**

C.1 Ablations 25

C.2 Analysis 26

C.3 Dense Task Specifications 31

A RecPo

A.1 Procedure Views

Procedure views provide a sparse way to govern how procedure records are read. To serve this role, a view should change when the factor it governs changes and remain stable when other factors vary. A view with this property exposes the local effects of skills without enumerating concrete task bindings or full skill sequences. In the main experiments, object- or factor-centric projections provide such views by selecting the observation dimensions of the governed factors. Other functions of the observation can provide such views as well, such as an image embedding for each camera in Appendix C.2.3 or the fill level of a cup in the real-world experiments of Appendix B.6.

Each view enters RecPo through its interface (α_v, δ_v), which determines how the procedure records are organized into skill-transition graphs and salience marks. Materialization is defined over this organized structure for an arbitrary view set rather than for a particular choice of views. The interface therefore separates representation from composition and serves as the basis of a modular materialization. Under this modularity, the same materialization rule applies whichever function instantiates the views, and changes to materialization in turn need not redefine how the views are obtained.

A.2 Procedure Knowledge Base

To compose stored skills through the interfaces of multiple procedure views, the procedure update U_Ω computes three kinds of evidence from the procedure substrate $(\mathcal{G}_{\Omega,\mathcal{V}}, \bar{\mathcal{X}}_{\Omega,\mathcal{V}})$ and the procedure records. The skill-transition graphs provide effect evidence on which view nodes a skill changes. The salience marks provide identification evidence on the view in which the effect of a skill separates it from the other stored skills. The segment endpoints and the skill sequences of the records provide order evidence on where a skill starts and ends and which skill follows it. None of this evidence depends on a task query. Throughout, let $\mathcal{I}_\omega = \{i : \omega_i = \omega\}$ denote the records of skill ω.

Effect evidence from the skill-transition graphs. For each view v, U_Ω canonicalizes the projected starts and endpoints of all records into the nodes $\mathcal{N}_v$. Records of different executions that reach the same view state then share a node, so the effect of a skill becomes a node change that can be counted over its records. U_Ω computes this partition with DBSCAN and assigns every stored point to a node. The map c_v extends the partition to a new state through the node of the nearest stored point, so task waypoints fall into the same nodes as the records. Over these nodes, U_Ω groups the edges of $\mathcal{G}_v$ in Eq. (6) by skill and start node. Let $\mathcal{I}_{\omega,v}(n) = \{i \in \mathcal{I}_\omega : c_v(z_{i,v}) = n\}$ denote the records of ω that start at node $n \in \mathcal{N}_v$. The empirical effect of ω at n is

$$\chi_v(n, \omega) = \mathbb{1}\left[\frac{\left|\{i \in \mathcal{I}_{\omega,v}(n) : c_v(z'_{i,v}) \neq c_v(z_{i,v})\}\right|}{|\mathcal{I}_{\omega,v}(n)|} \geq \frac{1}{2}\right], \tag{12}$$

which equals one when at least half of the records of ω that start at n change the node of view v. When no record of ω starts at n, all records of ω decide the effect. The view-change signature collects the views that ω changes in at least one record,

$$\Sigma_\omega = \{v \in \mathcal{V} : \exists i \in \mathcal{I}_\omega,\ c_v(z_{i,v}) \neq c_v(z'_{i,v})\}. \tag{13}$$

Identification evidence from the salience marks. For each view v, the salience mark measures how well the view-local effect of a record predicts its skill label. We instantiate the salience mark $m_{i,v}$ in Eq. (7) as the share of records with skill ω_i among the nearest neighbors of $e_{i,v}$ in view-local effect, where the neighbors include record i and the share excludes it. Averaging these marks over $\mathcal{I}_\omega$ gives the skill-view predictivity $\lambda_\omega(v)$, and we call $v^*_\omega \in \arg\max_v \lambda_\omega(v)$ the dominant view of ω. Task materialization reads the salience marks only through λ_ω.

Order evidence from the procedure records. For order evidence, U_Ω computes the initiation and termination centroids μ^s_ω and μ^e_ω from the procedure records without view projection. They are the means of the segment starts s_i and endpoints s'_i over $\mathcal{I}_\omega$. When the records come with their skill sequences, U_Ω also computes the empirical skill adjacency $\rightarrow_\Omega$ as a connectivity over the stored skills. It contains the ordered pairs (ω, ω') of distinct skills whose records are consecutive within one observed execution, independent of the context of that execution.

A.3 Task Materialization

Section 5.2 defines task-query abstraction and segment binding. Using the statistics in Appendix A.2, we specify below the support score, path objective, and algorithmic flow used by these stages. Table 1 lists where each statistic of Appendix A.2 enters these stages.

Table 1: Stored statistics of the Procedure Knowledge Base grouped by evidence and the materialization steps that read them. The last column names the ablation axis of Table 14 that changes how a statistic is used. †Read only when the records come with their skill sequences.

Evidence	Statistic	Used in task materialization	Ablation axis
effect	c_v	decision points, active views $\mathcal{V}_h$	–
	$\chi_v(n,\omega)$	admissible set Ω_h, decision-point refinement	–
	Σ_ω	selection of Ω_h^0, decision-point refinement	–
identification	λ_ω, v_ω^*	decision points, selection of Ω_h^0, projection $W_{\omega,h}$	view selection
order	μ_ω^s, μ_ω^e	candidate support ρ_h, path objective C_h, decision-point refinement	–
	$\rightarrow_\Omega$ †	final skill order	chaining prior

Decision-point refinement. A node change of Eq. (8) yields a decision point only when one changed view maximizes $\lambda_\omega(v)$ for at least one stored skill ω. The first and last steps of $\mathcal{Q}_\tau$ always remain decision points. A skill explains a segment when its signature Σ_ω covers the active views of the segment and its support ρ reaches ε_ρ, where ρ follows Eq. (14) with the active-view indicator of that segment in place of $W_{\omega,h}$. Among these skills, the explaining skill maximizes $\rho/|\Sigma_\omega|$. An interior decision point is removed only when a skill distinct from both adjacent explanations explains the combined segment and its support falls below the weaker adjacent support by at most the tolerance $\varepsilon_{\mathrm{drop}}$. Repeating this removal produces $\bar{\mathcal{Q}}_\tau$, and both settings of each ablation axis share it.

Candidate support. For a segment h of $\bar{\mathcal{Q}}_\tau$, let $\mathcal{V}_h$ be the active views and $\mathbb{1}_h$ the indicator of their observation dimensions defined in Section 5.2. Candidate support is

$$\rho_h(\omega) = \cos_{W_{\omega,h}}\left(q_{j_{h+1}} - q_{j_h}, \mu_\omega^e - \mu_\omega^s\right), \tag{14}$$

where $\cos_{W_{\omega,h}}$ is the cosine similarity after masking by $W_{\omega,h}$. The mask $W_{\omega,h}$ keeps a subset of the dimensions in $\mathbb{1}_h$, and its choice defines the view-selection axis of Table 14. Candidate-specific selection compares each candidate on its dominant view v_ω^*. When v_ω^* is inactive in the segment or $\lambda_\omega(v_\omega^*) = 0$, the comparison moves to the active views in which the stored records show one irreversible change, and to all of $\mathbb{1}_h$ when no such view is active. Uniform selection sets $W_{\omega,h} = \mathbb{1}_h$ for every candidate.

The admissible set Ω_h of segment h contains the skills whose empirical effect in Eq. (12) changes the start node of at least one active view. When the signature Σ_ω of some candidate covers $\mathcal{V}_h$, both settings consider only the candidates with such a signature. Otherwise, candidate-specific selection keeps only the candidates whose dominant view is active with $\lambda_\omega(v_\omega^*) > 0$, and uniform selection keeps all of them. The selected candidates form Ω_h^0 and satisfy $\rho_h(\omega) \geq \varepsilon_\rho$ for the support threshold ε_ρ.

Path objective. The path operator in Eq. (10) orders the skills of Ω_h^0 by approximately minimizing

$$C_h(\omega_{1:L_h}) = d_h(q_{j_h}, \mu_{\omega_1}^s) + \sum_{l=1}^{L_h-1} d_h(\mu_{\omega_l}^e, \mu_{\omega_{l+1}}^s) + d_h(\mu_{\omega_{L_h}}^e, q_{j_{h+1}}), \tag{15}$$

where d_h is the L_2 distance on the dimensions of $\mathbb{1}_h$. A bounded bidirectional beam search expands partial sequences from both endpoints and orders every skill in Ω_h^0, so $L_h = |\Omega_h^0|$.

Composition and chaining prior. Task-level composition emits each selected skill once at its earliest segment binding and returns one executable skill chain. When the records come with their skill sequences, the empirical skill adjacency $\rightarrow_\Omega$ reorders this chain only when it admits exactly one order that covers all chained skills. The reorder applies to the whole chain and can cross segment boundaries. It defines the chaining-prior axis of Table 14, and chaining without the prior keeps the composed order.

The simulation protocol executes the fixed sequence returned by Algorithm 1. This design isolates task abstraction and composition from feedback control. The reported results therefore characterize

Algorithm 1 Task materialization in `RecPo`

Require: task query $\mathcal{Q}_\tau$ and Procedure Knowledge Base $\mathcal{K}_\Omega$
Ensure: materialized task $\widehat{\mathcal{Q}}_\tau$

▷ Abstract the task query
Project $\mathcal{Q}_\tau$ through the supplied views and collect node changes containing a view that maximizes $\lambda_\omega(v)$ for a stored skill.
Refine the decision points with stored segment support to form $\bar{\mathcal{Q}}_\tau$.

▷ Bind the abstracted segments
for each segment $(q_{j_h}, q_{j_{h+1}})$ in $\bar{\mathcal{Q}}_\tau$ **do**
  Identify the active views $\mathcal{V}_h$ and indicator $\mathbb{1}_h$.
  Collect the skills with $\chi_v = 1$ at the active-view start nodes.
  Select Ω_h^0 from the admissible set Ω_h satisfying $\rho_h(\omega) \geq \varepsilon_\rho$.
  Bind the segment to $\hat{\omega}_{h,1:L_h}$ by approximately minimizing C_h in Eq. (15) over Ω_h^0.

▷ Compose the executable task
Emit every selected skill once at its earliest segment binding.
Reorder these bindings only if $\rightarrow_\Omega$ defines a unique covering order.

return $\widehat{\mathcal{Q}}_\tau$ as defined in Eq. (11).

materialization before execution-layer disturbance detection or recovery is introduced. Table 2 lists the hyperparameters of the Procedure Knowledge Base and task materialization.

Table 2: Hyperparameters of the Procedure Knowledge Base and task materialization.

Stage	Hyperparameter	Value
Procedure Knowledge Base	DBSCAN neighborhood radius	0.10
	DBSCAN minimum sample count	20
	salience nearest neighbors	32
Task materialization	support threshold ε_ρ	0.10
	refinement tolerance $\varepsilon_{\mathrm{drop}}$	0.05
	beam width	8

B Experiment Details

For the camera-ready version, we update the results by extending the evaluation scope of the experiments that were limited in the submitted version and the rebuttal, and by re-measuring task-inference times with a codebase that uses optimized clustering and nearest-neighbor retrieval algorithms.

B.1 Environment

Procedure records. We populate the procedure knowledge base with the length-4 Franka Kitchen realizations provided by [58, 59]. Each realization starts from the common initial configuration and executes four of seven subtasks in the relative order of the target task in Table 5. Because the realizations instantiate different subtask subsets, the same skill may be executed after different preceding subtasks have been completed, yielding distinct full-state contexts for the same view-local effect. We refer to this variation in the contexts supporting reusable skill effects as *skill-context diversity*.

Held-out length-7. The held-out length-7 target completes all seven subtasks, while its complete realization is excluded from the procedure knowledge base. Figure 7 shows the common initial configuration and the state after all seven subtasks are completed. It therefore requires skill-induced effects observed in the length-4 records to be reused across joint contexts absent from any stored realization. This shared relative order constrains the subtask order, leaving task materialization to identify transition evidence for the demanded effects and compose the corresponding skills. We further increase the *waypoint gap* H to remove intermediate task-state observations without changing this ordering structure. As H grows, a single observed waypoint transition may span multiple subtask effects, reducing the task-side evidence available for binding them to stored procedure records.

The held-out length-7 setting thus evaluates composition of reusable effects across unseen context combinations under increasingly sparse task evidence.

KB construction details. A procedure realization is a sequence of 30-dimensional joint configurations. Each configuration holds the 9 robot joints (7 arm joints and 2 gripper fingers) and the 21 object joints. Each subtask has a goal configuration of its object joints [59], and a subtask is complete when its object joints are within the Euclidean distance 0.3 of this goal. Each observation has the label of the subtask that the robot executes. The label changes after the subtask is complete and its object stops moving. We split each realization into maximal runs of the same label. Each run gives one procedure record $x_i = (s_i, s'_i, \omega_i, k_i)$ in Eq. (4). Thus a record starts where the previous record ends, and its end observation shows the settled effect of its skill. The motion of the robot toward the next object belongs to the next record. We use the labels only for this split. The knowledge base does not receive a task identity or a target order.

We group the realizations by their length-4 subtask order (Table 3). The tables name each subtask by its object as `MW` for the microwave, `KT` for the kettle, `BB` for the bottom burner, `TB` for the top burner, `LS` for the light switch, `SC` for the slide cabinet, and `HC` for the hinge cabinet. 24 orders come from the original Franka Kitchen demonstrations [58], and we add 8 orders. We build the realizations of the added orders by stitching subtask segments of the original demonstrations. Each order gives 18 to 25 realizations. A skill-context diversity setting K contains the orders with the IDs 1 to K. Thus a larger K only adds records to a smaller K.

- $K = 4$: 93 realizations, 372 procedure records.
- $K = 8$: 193 realizations, 772 procedure records.
- $K = 16$: 393 realizations, 1,572 procedure records.
- $K = 32$: 776 realizations, 3,104 procedure records.

We build the knowledge base one time for each pair of K and view configuration. All waypoint gaps, seeds, and evaluation episodes use the same knowledge base.

Table 3: Length-4 subtask orders of the procedure realizations. A number gives the step of the subtask in the order. Unshaded rows are the 24 orders of the original Franka Kitchen demonstrations [58], and shaded rows are the 8 added orders.

ID	MW	KT	BB	TB	LS	SC	HC	ID	MW	KT	BB	TB	LS	SC	HC
1	·	·	1	·	2	3	4	17	·	1	·	2	3	·	4
2	1	2	·	3	·	·	4	18	1	·	2	·	3	4	·
3	·	·	1	2	3	4	·	19	·	1	·	2	·	3	4
4	1	2	3	·	·	·	4	20	1	·	2	·	·	3	4
5	·	1	·	2	3	4	·	21	·	1	2	·	3	4	·
6	1	·	2	·	3	·	4	22	1	·	·	2	3	·	4
7	1	2	·	3	·	4	·	23	·	1	2	·	·	3	4
8	·	·	1	2	3	·	4	24	1	·	2	3	4	·	·
9	1	2	3	·	·	4	·	25	1	2	·	·	3	·	4
10	·	1	·	·	2	3	4	26	·	1	2	3	·	·	4
11	1	·	2	3	·	·	4	27	1	·	·	2	3	4	·
12	1	2	·	·	3	4	·	28	·	1	2	3	4	·	·
13	·	·	1	2	·	3	4	29	1	2	·	·	·	3	4
14	1	2	·	3	4	·	·	30	1	·	·	2	·	3	4
15	·	1	2	·	3	·	4	31	·	1	2	3	·	4	·
16	1	·	2	3	·	4	·	32	1	·	·	·	2	3	4

The procedure views follow the object factors that D4RL defines for the Franka Kitchen state [59]. Each factor is a fixed set of state dimensions, and Table 4 gives these dimensions. These factors are not privileged information. Counterfactual tests on interaction data can identify the state factors that an action or an entity changes [62, 60, 63]. Every configuration keeps the robot view and the kettle view, and it merges the other six object factors into $V - 2$ views.

Task specification. The task demonstration is one held-out length-7 realization with 383 observations. Its subtask order is the target order in Table 5. We split it with the same rule as the procedure realizations. The initial observation and the end observations of the seven runs give eight boundary

Table 4: State dimensions of the view factors (top) and views of the seven view configurations (bottom). RB is the robot. The robot view also holds the dimensions 9–10 of a burner that no subtask uses. The dimensions 13–14 of the other unused burner are in no view.

Factor	RB	BB	TB	LS	SC	HC	MW	KT
Dimensions	0–10	11–12	15–16	17–18	19	20–21	22	23–29

V	Fixed views	Merged object views
8	{RB} {KT}	{MW} {BB} {TB} {LS} {SC} {HC}
5	{RB} {KT}	{LS, SC} {MW, HC} {BB, TB}
5	{RB} {KT}	{BB, HC} {TB, LS} {MW, SC}
5	{RB} {KT}	{MW, SC} {LS, HC} {BB, TB}
4	{RB} {KT}	{MW, LS, SC, HC} {BB, TB}
4	{RB} {KT}	{BB, TB, LS, HC} {MW, SC}
3	{RB} {KT}	{MW, BB, TB, LS, SC, HC}

observations with the indices $\{0, \ldots, 7\}$. For a waypoint gap H, the task record keeps the boundary indices $\{0, H, 2H, \ldots\}$ that are not larger than 7. If the last kept index is not 7, the task record also keeps the final observation. At $H = 7$, only the initial and the final observations remain. All methods receive the same waypoint array, and its first row is the initial observation of the episode. No method receives subtask labels or skill identities from the task demonstration.

Table 5: Subtask order of the held-out length-7 target task.

Step	1	2	3	4	5	6	7
Subtask	MW	KT	BB	TB	LS	SC	HC

Executable skill construction. The evaluation executes a materialized skill sequence with a library of executable skills, one skill for each kitchen object. A skill is one kinematic transition. It sets the joints of its object to the goal configuration of its subtask and sets the 9 robot joints to a robot pose sampled from the end observations of the records of the same skill in the length-4 realizations. All other joints keep their values, so the outcome of an episode depends only on the selected skills and their order. After each skill, the environment returns the joint configuration with the sensor noise of Franka Kitchen. The evaluation thus measures procedure reconfiguration.

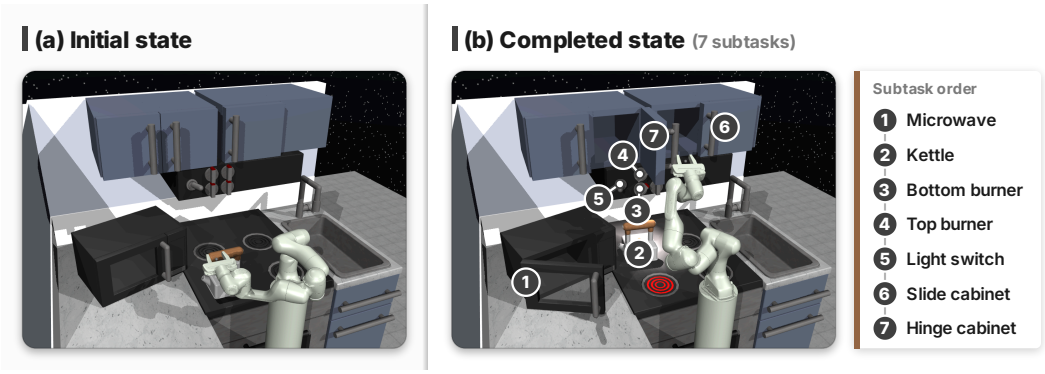


Figure 7: Franka Kitchen environment. (a) The common initial configuration from which every realization starts. (b) The state after the held-out length-7 target completes all seven subtasks, numbered in the target order.

B.2 Baseline Implementations

All baselines use the same stored skill records, supplied procedure views, and executable skill library.

`Skill Retrieval (SR).` SR performs transition-level retrieval and reranking. For each task-waypoint transition, it scores each skill by matching the requested transition to the closest stored record of that skill under each supplied procedure view. The view-wise scores are averaged to obtain a single retrieval score for each skill, and the highest-scoring skill is selected independently for each waypoint transition.

`Skill Sequence Retrieval (SSR).` SSR extends the retrieval unit from individual skill records to stored skill subsequences. For each length-4 procedure realization, we construct retrieval items from its contiguous skill subsequences of lengths 1, 2, and 3, preserving their observed execution order. Each subsequence is represented under the same supplied procedure views as SR and matched to a task-waypoint segment using the same multi-view scoring criterion. For each task segment, SSR retrieves the highest-scoring stored subsequence. This baseline evaluates whether retrieval over previously observed short skill subsequences is sufficient to recover the target behavior from sparse task evidence.

`Skill Chaining (SC).` SC instantiates the structured procedural-representation family of skill chaining and skill-graph methods [46, 48] within the shared reconfiguration interface above. It uses the supplied procedure views when matching task waypoints to stored record endpoints, giving it access to the same factorized procedure information as RecPo.

The distinction lies in how this evidence is organized for composition. SC summarizes the records associated with each skill into canonical initiation and termination states in the procedure-view space, thereby representing procedural connectivity at the skill level. Given a task query, it scores candidate skills by agreement with the inferred waypoint changes and constructs a skill-transition graph whose edges are weighted by termination-to-initiation distance. A bounded path search over this graph returns the executable skill sequence. SC therefore tests whether the supplied factorized procedure information remains sufficient after record-level evidence is consolidated into skill-level connectivity. In contrast, RecPo retains record-derived view-local transition evidence and resolves the task-relevant composition during materialization.

B.3 Computing Details

Table 6 lists the compute resources and core library versions. RecPo uses NumPy for numerical array operations and RAPIDS-based GPU libraries for scalable clustering and nearest-neighbor retrieval. We use scikit-learn and UMAP for CPU-based clustering and dimensionality reduction.

Table 6: Compute resources and core library versions.

Compute resources		**Clustering and numerical libraries**			
CPU	AMD Ryzen Threadripper PRO 9975WX (32-core)	`numpy`	2.2.6	`cuml`	25.12.0
GPU	2× NVIDIA GeForce RTX 5090	`scipy`	1.16.3	`cuvs`	25.12.0
OS	Ubuntu 24.04.3 LTS	`scikit-learn`	1.7.2	`cugraph`	25.12.2
CUDA	12.8	`umap-learn`	0.5.11	`cupy`	14.0.1
Driver	570.211.01	`pynndescent`	0.6.0		

B.4 Evaluation Protocol and Aggregation

The procedure-support and task-record-sparsity analyses use the sweep in Table 8. The stored length-4 realizations provide the skill-context diversity settings. The seven procedure-view configurations use distinct factor-merging assignments under the same view-construction rule. The $V = 3, 4, 5, 8$ levels contain $1, 2, 3, 1$ configurations, respectively. Both analyses report marginals of this sweep. The task-input conditions are described in Section B.2.

Experiment condition tables use the notation in Table 7. Counts cover the listed configurations for each method and experimental variant. For randomized view construction, we first average within each construction seed. We then report the mean and sample standard deviation across construction seeds.

Ordered-completion evaluation. An episode ends after at most 12 skill executions, and ordered completion (OC) counts the subtasks completed in the target order. Each configuration uses $S_e = 10$

Table 7: Notation for experiment condition tables.

Notation	Meaning
K, H, V	Skill-context diversity, waypoint gap, and view resolution.
S_e	Number of evaluation seeds, each with 100 episodes.
N_{cfg}	Number of listed configurations.
N_{ep}	Total quality-evaluation episodes, $100\, N_{\text{cfg}} S_e$, with $\text{k} = 10^3$.
†	An axis averaged over its listed configurations. Additional averaging for Mean is stated in each caption.
–	Not applicable.

Table 8: 196-configuration sweep for procedure-support and task-record-sparsity evaluation. N_{ep} counts quality-evaluation episodes per method.

K	H	Views	N_{cfg}	N_{ep}
$\{4, 8, 16, 32\}$	$\{1, \ldots, 7\}$	$V \in \{3, 4, 5, 8\}$ 7 configurations total	$4 \times 7 \times 7$	196k

evaluation seeds and 100 episodes per seed, yielding 196,000 episodes per method. We first average all episodes across the ten seeds within each configuration and then weight configurations equally. Marginal results fix one axis and average over the remaining configurations. Accordingly, each K value averages 49 configurations, each H value averages 28, and the four V levels average 28, 56, 84, and 28 configurations, respectively.

Timing evaluation. Timing is measured in a separate run over the same configurations. Each configuration uses two seeds and 100 repetitions per seed. We exclude the first repetition of each seed as warmup, leaving 198 measurements per configuration and 38,808 measurements per method. Materialization is recomputed for every repetition, while the procedure knowledge base is reused. The reported task-inference time includes task-query abstraction, materialization, and all policy-prediction calls within an episode, while procedure-knowledge-base construction, file loading, environment communication, and physical execution are excluded. Quality and timing use the same configuration weighting while being measured independently.

Aggregation and variability. For the sweep above, error terms for both *ordered completion* and timing are sample standard deviations across configuration means. Marginals hold the displayed axis fixed and vary the remaining configuration dimensions, including distinct view configurations at the same V. These deviations describe configuration heterogeneity and are not confidence intervals.

B.5 Main Evaluation Results

Table 9 summarizes *ordered completion* and materialization and prediction time over the sweep in Table 8. The following subsections detail *procedure support* and *task-record sparsity* separately.

B.5.1 Procedure-Support Evaluation

We examine *procedure support* by varying *skill-context diversity* and *view resolution*. The marginal results below show how completion and computation vary with *procedure support*.

Skill-context diversity. Table 10 holds K fixed while averaging all supplied view configurations and *waypoint gaps*. It therefore measures how performance changes as more length-4 realizations provide *procedure support* for the same held-out target.

For `RecPo`, *ordered completion* increases from 81.0% at $K = 4$ to 97.7% at $K = 32$, while time increases from 0.5 to 2.0 ms. The completion gains become smaller at larger K even as computation continues to increase. The baselines do not show the same monotonic completion trend. More stored realizations therefore accompany improved completion for `RecPo` in this sweep, but do not produce a uniform benefit across the evaluated methods.

Table 9: Mean *ordered completion* and materialization and prediction time over the procedure-support and task-record-sparsity sweep. Error terms describe variation across configuration means under Section B.4.

Experiment Config				
$K^{\dagger}$	$H^{\dagger}$	Views†	N_{cfg}	N_{ep}
$\{4, 8, 16, 32\}$	$\{1, \ldots, 7\}$	$V \in \{3, 4, 5, 8\}$ 7 configurations total	$4 \times 7 \times 7$	196k

Results

	`Skill Retrieval (SR)`		`Skill Sequence Retrieval (SSR)`		`Skill Chaining (SC)`		`RecPo (ours)`	
Average	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)
Overall	8.1±15.9	2.4±1.0	24.7±29.5	6.7±2.0	51.8±34.3	0.9±0.7	91.9±15.9	1.0±0.7

Table 10: *Ordered completion* and materialization and prediction time by *skill-context diversity* K. Both error terms describe variation across configuration means.

Experiment Config				
K	$H^{\dagger}$	Views†	N_{cfg}	N_{ep}
$\{4, 8, 16, 32\}$	$\{1, \ldots, 7\}$	$V \in \{3, 4, 5, 8\}$ 7 configurations total	$4 \times 7 \times 7$	196k

Results

	`Skill Retrieval (SR)`		`Skill Sequence Retrieval (SSR)`		`Skill Chaining (SC)`		`RecPo (ours)`	
K	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)
4	9.6±15.9	1.6±0.5	20.5±23.8	5.3±1.6	56.9±32.0	0.5±0.2	81.0±21.1	0.5±0.2
8	7.1±14.4	1.9±0.6	18.0±23.0	6.5±1.8	50.7±36.0	0.7±0.2	92.7±14.6	0.7±0.2
16	7.1±14.6	2.5±0.6	21.6±33.7	6.9±1.9	52.2±33.6	0.8±0.3	96.2±12.3	0.9±0.4
32	8.7±18.7	3.6±0.9	38.7±32.1	7.9±2.0	47.5±35.8	1.8±0.8	97.7±6.7	2.0±0.8

View resolution. Table 11 fixes the number of supplied views and averages *skill-context diversity*, *waypoint gaps*, and all configurations at that resolution.

`RecPo` rises from 73.5% *ordered completion* at $V = 3$ to 100.0% at the supplied $V = 8$ configuration. Its corresponding time increases from 0.8 to 1.3 ms. `SC` also reaches its highest marginal completion at $V = 8$, whereas neither retrieval baseline improves monotonically with view count. These trends concern the supplied configurations, in which view count and partition structure vary together.

B.5.2 Task-Record Sparsity Evaluation

We next fix the *waypoint gap* and average *procedure support* under Section B.4. Increasing H leaves fewer intermediate task waypoints while preserving the held-out target and procedure records. Table 12 reports the resulting *ordered completion* and materialization and prediction time. Its ordered-completion deviations describe heterogeneity across the 28 configurations at each gap.

Table 11: *Ordered completion* and materialization and prediction time by supplied *view resolution* V. Both error terms describe variation across configuration means.

Experiment Config				
$K^\dagger$	$H^\dagger$	Views	N_{cfg}	N_{ep}
$\{4, 8, 16, 32\}$	$\{1, \ldots, 7\}$	$V = 3$ 1 configuration	$4 \times 1 \times 7$	28k
$\{4, 8, 16, 32\}$	$\{1, \ldots, 7\}$	$V = 4$ 2 configurations†	$4 \times 2 \times 7$	56k
$\{4, 8, 16, 32\}$	$\{1, \ldots, 7\}$	$V = 5$ 3 configurations†	$4 \times 3 \times 7$	84k
$\{4, 8, 16, 32\}$	$\{1, \ldots, 7\}$	$V = 8$ 1 configuration	$4 \times 1 \times 7$	28k

Results

	`Skill Retrieval (SR)`		`Skill Sequence Retrieval (SSR)`		`Skill Chaining (SC)`		`RecPo (ours)`	
V	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)
3	7.4±16.5	1.7±0.6	18.5±28.6	5.0±1.0	40.8±26.9	0.7±0.5	73.5±21.2	0.8±0.5
4	8.7±15.9	2.1±0.8	27.1±29.7	5.8±1.2	40.1±30.9	0.9±0.6	90.6±17.5	0.9±0.6
5	8.4±15.9	2.5±0.9	25.3±29.6	6.8±1.6	49.3±31.4	1.0±0.7	96.3±9.4	1.0±0.7
8	7.0±16.1	3.5±1.1	24.3±30.4	9.7±2.1	93.9±22.8	1.2±1.0	100.0±0.0	1.3±1.0

Table 12: *Ordered completion* and materialization and prediction time by *waypoint gap* H. Both error terms describe variation across configuration means.

Experiment Config				
$K^\dagger$	H	Views†	N_{cfg}	N_{ep}
$\{4, 8, 16, 32\}$	$\{1, \ldots, 7\}$	$V \in \{3, 4, 5, 8\}$ 7 configurations total	$4 \times 7 \times 7$	196k

Results

	`Skill Retrieval (SR)`		`Skill Sequence Retrieval (SSR)`		`Skill Chaining (SC)`		`RecPo (ours)`	
H	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)
1	43.8±5.7	1.7±0.6	68.7±17.2	8.1±2.0	36.2±35.1	1.6±1.1	92.3±15.8	1.8±1.1
2	6.6±12.0	2.0±0.8	46.1±29.3	7.5±1.9	51.5±34.2	1.1±0.7	89.8±17.8	1.2±0.7
3	0.0±0.0	2.6±1.0	37.6±19.8	7.2±1.8	53.6±27.1	1.0±0.6	89.3±19.3	1.0±0.6
4	0.0±0.0	2.7±1.0	9.1±16.1	5.4±1.3	58.7±26.8	0.7±0.5	89.8±17.8	0.8±0.5
5	3.2±6.3	2.5±1.0	6.6±11.4	6.1±2.0	53.6±32.9	0.8±0.5	91.8±14.8	0.8±0.5
6	3.5±6.2	2.5±1.1	1.3±4.1	6.7±2.1	39.3±29.8	0.8±0.5	95.4±12.3	0.8±0.5
7	0.0±0.0	2.7±1.0	3.5±7.2	5.5±1.7	69.9±43.0	0.6±0.3	94.9±13.0	0.7±0.4

B.5.3 Procedure Knowledge Base Construction

To assess how procedure knowledge base preparation scales with skill-context diversity, we measure its one-time construction cost in Table 13. This measurement includes loading the procedure records and constructing the representation used by each method. For `RecPo`, it includes constructing the marked records and procedure substrate. We reuse the constructed knowledge base across task evaluations.

Table 13: Mean procedure knowledge base preparation time by skill-context diversity, in ms. Each value averages five warm construction measurements for each of seven view configurations, excluding the first construction in each process.

Experiment Config				
K	H	Views†	N_{cfg}	N_{ep}
$\{4, 8, 16, 32\}$	–	$V \in \{3, 4, 5, 8\}$ 7 configurations total	4×7	–

Results				
K	SR	SSR	SC	RecPo
4	2.7	10.3	18.6	22.1
8	5.3	21.6	33.5	33.5
16	11.6	38.5	64.8	63.6
32	24.4	85.4	140.4	138.0

Preparation time increases with skill-context diversity for every method. For RecPo, it ranges from 22.1 to 138.0 ms across the evaluated settings. The first construction in each process additionally takes 1.22 s for RecPo and 1.06 s for SC, mostly for accelerator-library initialization, and 13 ms and 25 ms for SR and SSR; it is excluded from Table 13. This preparation cost is excluded from the task-side computation reported in Section B.4.

B.6 Real-World Evaluation

Workspace configuration. We evaluate RecPo in a cafe workspace centered on coffee preparation. We deploy across two environments that share the goal (*pour coffee*) but differ in scene composition (Figure 8), so that procedure records collected under one scene composition are reused under another.

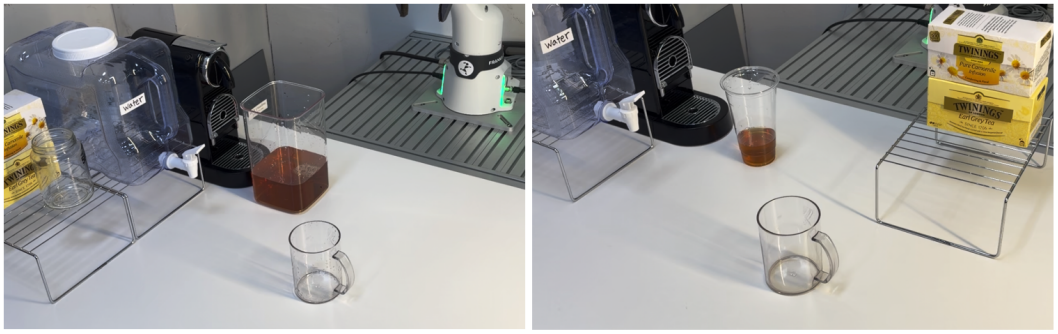


Figure 8: The two physical environments used for real-world evaluation, sharing the goal *pour coffee* but differing in scene composition. (***left***) Environment 1: a large source whose volume exceeds the target cup capacity. (***right***) Environment 2: a source whose volume matches the target cup capacity.

Robots. The workspace is equipped with a 7-DoF Franka Emika Research 3 tabletop manipulator.

Perception. The workspace is equipped with two cameras serving distinct roles. An Intel RealSense D435 RGB-D camera mounted top-down provides object-level grounding: RGB frames are passed to an object detector that returns category labels and bounding boxes for task-relevant objects, while the aligned depth channel lifts each detection to a 3D position expressed in the arm's base frame via the camera-to-robot extrinsics. A separate front-view RGB camera supplies the visual input read by the partial procedure views $\mathcal{V}$, on which the projection maps α_v are defined.

Low-level controller. Skill executions are realized as Cartesian waypoint sequences in the arm's base frame and dispatched through MoveIt, which handles inverse kinematics and trajectory planning to the underlying joint controllers.

B.6.1 Reconfiguration Scenarios

Both environments share the goal *pour coffee*. They differ in the source-to-target capacity relation and in the procedure records available at deployment. The supplied views represent the source and target factors needed for the pouring and fill-level transitions. We compare how local updates to $\mathcal{K}_\Omega$ affect `RecPo` and `SC` under the same memory.

Case 1: interference from other views. The source contains more liquid than the target can hold. The task therefore requires a stopping condition before the source becomes empty. The default views expose the stored pouring transition through the source state but do not distinguish the target fill level. We add a supplied target-fill view and a procedure record for the stopping transition. `RecPo` then composes the pouring and stopping skills from their view-local effects. `SC` reuses the pouring skill from endpoint compatibility alone and overfills the target.

Case 2: recovery through records. The source volume matches the target capacity. The updated pouring segment requires one skill ($L_h{=}1$), while the full task contains two ordered skills. The memory also contains endpoint-compatible records whose target-side effects do not match the requested fill transition. `SC` can therefore select an incorrect record from endpoint compatibility. We append the correct pouring record without changing the skill library. Its target-view transition matches the requested effect, so `RecPo` binds the correct pouring skill within the two-skill task.

B.6.2 Record-Appending Scenario

For the second scenario above, procedure records are collected under three scene conditions. These collection conditions are distinct from the two physical reconfiguration scenarios. The evaluation combines background and manipulated-object conditions not paired during collection. We run ten trials per method under the same lighting and nominal object placement. Sequence success requires the ordered two-skill sequence from the eight-skill library. Task success additionally requires completing the pour without spilling. Over the ten trials per method, `RecPo` achieves $10/10$ sequence success and $7/10$ task success, whereas `SC` achieves $0/10$ on both.

The three failed `RecPo` trials use the correct materialized sequence and are attributed to grasp, alignment, or depth-localization errors.

C Additional Experiments

Supplementary evaluation. To evaluate supplementary sensitivity, we use the experiment-specific configurations listed with each result table. Evaluation and aggregation follow Section B.4.

C.1 Ablations

Table 14 removes one kind of evidence of Appendix A.2 at a time from task materialization, with the supplied views fixed. We use the sweep in Table 8 and the aggregation in Section B.4. Both axes leave the decision-point refinement of Appendix A.3 unchanged.

- **View selection.** Candidate-specific selection uses the identification evidence to choose the mask $W_{\omega,h}$ and the candidates of Ω_h^0, as specified in Appendix A.3. Uniform selection removes this evidence from segment binding. It sets $W_{\omega,h} = \mathbb{1}_h$ for every candidate and keeps every candidate when no signature covers $\mathcal{V}_h$.
- **Chaining prior.** Both settings keep the path objective of Eq. (15). *Prior-guided chaining* reorders the composed chain with the empirical skill adjacency $\rightarrow_\Omega$ when it admits exactly one order covering the chained skills. *Chaining* keeps the composed order and uses no skill order of the procedure records.

All other `RecPo` settings are the same. The skill order that $\rightarrow_\Omega$ reads is the same evidence that `SSR` retrieves, and the chaining setting uses no more order evidence than `SC`.

We also include `SC`, which shares the decision-point refinement of `RecPo` and orders its candidates by the same path objective over the centroids. Its candidate selection differs from the uniform chaining setting of `RecPo`. `SC` does not restrict the candidates by signature. It keeps the condition that the

dominant view of a candidate is active. It keeps every candidate with positive support instead of applying ε_ρ.

Table 14: Mean OC for the view-selection and chaining-prior ablations of RecPo and the SC comparison on the sweep in Table 8. Configuration and episode counts are per row. The default RecPo result is bold.

Experiment Config				
$K^\dagger$	$H^\dagger$	Views†	N_{cfg}	N_{ep}
$\{4, 8, 16, 32\}$	$\{1, \ldots, 7\}$	$V \in \{3, 4, 5, 8\}$ 7 configurations total	$4 \times 7 \times 7$	196k

Results			
Method	View selection	Chaining prior	OC↑(%)
RecPo	Candidate-specific	Prior-guided chaining	**91.9**
		Chaining	86.3
	Uniform	Prior-guided chaining	89.1
		Chaining	70.6
SC	–	–	51.8

Removing either kind of evidence lowers OC. Without the chaining prior, uniform selection lowers OC from 86.3% to 70.6%, while with the prior it lowers OC only from 91.9% to 89.1%.

C.2 Analysis

C.2.1 Robustness to Procedure-View Specification

To assess sensitivity to procedure-view specification, we vary partition structure and overlap under the supplementary protocol in Section C. Both reference and random assignments use separate view-construction seeds while keeping the underlying observations fixed. The partitioning seeds for random assignments are distinct from the evaluation seeds. This reference distribution differs from the fixed views used in the main evaluation.

To examine the role of the supplied partition structure, we redistribute the same observation dimensions into random partitions with the same view count. Table 15 compares these partitions with the reference object-factor assignments.

Table 15: OC under reference and random procedure-view partitions. S_p denotes the number of view-construction seeds, distinct from evaluation seeds. For random partitions, each seed determines the assignment of observation dimensions to views. Under random partitions, RecPo tuned canonicalization for random partitions.

Experiment Config					
$K^\dagger$	$H^\dagger$	Views†	$S_p^\dagger$	N_{cfg}	N_{ep}
$\{4, 8, 16, 32\}$	$\{1, 2, 4, 7\}$	$V \in \{3, 4, 5, 8\}$	5	$4 \times 4 \times 4 \times 5$	320k

Results		
Method	Reference views	Random partitions
SR	12.5 ± 0.2	13.1 ± 1.1
SSR	31.2 ± 0.5	21.3 ± 1.1
SC	55.2 ± 1.9	34.9 ± 6.4
RecPo	90.5 ± 2.4	63.2 ± 8.3

Random partitions lower the OC of RecPo, SC, and SSR and widen the variation across construction seeds, while SR remains near its low reference level. RecPo retains the highest OC under both assignments. The methods that compose skills over the supplied views thus depend on how the same observation dimensions are assigned to views.

To measure sensitivity to overlapping views, we copy a varying fraction of factors into another existing view, as reported in Table 16. The number of views, underlying observations, and remaining evaluation settings are unchanged. At duplication fraction 1, every factor appears in two views.

Table 16: OC as the factor-duplication fraction varies. S_p denotes the number of view-construction seeds, distinct from evaluation seeds.

Experiment Config					
$K^\dagger$	$H^\dagger$	Views†	$S_p^\dagger$	N_{cfg}	N_{ep}
$\{4, 8, 16, 32\}$	$\{1, 2, 4, 7\}$	$V \in \{3, 4, 5, 8\}$	5	$4 \times 4 \times 4 \times 5$	320k

Results				
Duplication fraction	0	0.25	0.5	1.0
SR	12.5 ± 0.2	14.7 ± 1.6	14.7 ± 2.9	11.6 ± 0.5
SSR	31.2 ± 0.5	36.0 ± 10.8	35.2 ± 8.7	27.3 ± 2.7
SC	55.2 ± 1.9	49.1 ± 7.9	47.7 ± 5.5	33.6 ± 4.7
RecPo	90.5 ± 2.4	68.0 ± 12.4	65.7 ± 8.5	49.7 ± 13.1

As the duplication fraction increases, the OC of RecPo and SC decreases, with the largest drop when every factor is duplicated. SSR and SR vary within a narrower range around their low reference levels. RecPo retains the highest OC at every duplication fraction. Overlapping views therefore degrade the methods that compose skills through view-level structure, and duplicating observations across views does not preserve the performance of the reference interface.

C.2.2 Canonicalization Sensitivity

To assess how endpoint canonicalization affects completion, we vary the canonicalization map while holding the remaining materialization settings fixed on the sweep in Table 8. Table 17 compares DBSCAN with connected components of an ϵ-neighborhood graph and KMeans with its cluster count selected by the silhouette score. The neighborhood radius is 0.1 for DBSCAN and the ϵ-neighborhood graph.

Table 17: RecPo OC across endpoint canonicalization procedures.

Experiment Config				
$K^\dagger$	$H^\dagger$	Views†	N_{cfg}	N_{ep}
$\{4, 8, 16, 32\}$	$\{1, \ldots, 7\}$	$V \in \{3, 4, 5, 8\}$ 7 configurations total	$4 \times 7 \times 7$	196k

Results			
Metric	Canonicalization algorithm		
	DBSCAN	ϵ-neighborhood graph	Silhouette KMeans
OC↑(%)	91.9	88.0	88.3

Mean OC ranges from 88.0% to 91.9% across the three procedures. DBSCAN attains the highest mean in this comparison.

To assess sensitivity to DBSCAN parameters, we evaluate all 54 combinations of nine neighborhood radii and six minimum sample counts in Table 18. The default remains radius 0.10 with minimum sample count 20.

Table 18: `RecPo` OC across DBSCAN parameters. The bold entry denotes the default setting.

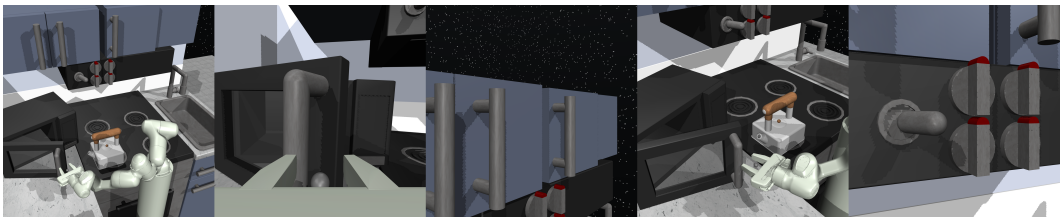
Experiment Config				
$K^\dagger$	$H^\dagger$	Views†	N_{cfg}	N_{ep}
$\{4, 8, 16, 32\}$	$\{1, \ldots, 7\}$	$V \in \{3, 4, 5, 8\}$ 7 configurations total	$4 \times 7 \times 7$	196k

Results						
Neighborhood radius ϵ	Minimum sample count					
	1	5	10	20	40	80
0.010	33.4	33.6	24.0	14.8	18.9	11.7
0.025	51.8	68.3	71.1	42.8	45.6	25.2
0.050	78.8	93.7	90.0	85.1	85.9	63.3
0.075	88.0	94.1	93.8	88.3	83.1	74.5
0.100	88.0	94.4	95.6	**91.9**	85.7	75.5
0.150	88.0	94.1	97.2	96.1	87.8	79.2
0.200	97.2	97.2	97.2	97.2	92.3	83.2
0.300	14.3	14.3	14.3	14.3	14.3	14.3
0.500	8.2	8.2	8.2	8.2	8.2	7.8

Across the evaluated combinations, mean OC ranges from 7.8% to 97.2%. For radii from 0.05 to 0.20 and minimum sample counts from 5 to 20, it lies between 85.1% and 97.2%. Five combinations attain the maximum, while the larger radii 0.30 and 0.50 yield substantially lower completion.

C.2.3 Visual Procedure Views

To assess whether `RecPo` can compose skills from visual observations, we define one procedure view per camera using DINOv3 [64] image embeddings. These views exclude sensor-state features. Figure 9 illustrates the five camera observations of the same Kitchen state. We vary the number of cameras from one to five, using the first n cameras shown from left to right for each camera count n.



(a) Overview (b) Wrist (c) Cabinets (d) Stovetop (e) Controls

Figure 9: Five camera views of the same Kitchen state, rendered at higher resolution for visualization. Each image supplies a separate visual procedure view.

To examine the effect of camera coverage, we compare the five camera subsets in Table 19. `RecPo` constructs a skill sequence before execution and does not replan from new camera observations during execution. Table 19 reports OC for each camera count, averaging over skill-context diversity and waypoint gap.

`RecPo` attains 87.0% mean OC across the five camera subsets. The highest camera-count mean is 92.9% with four cameras, and performance does not increase monotonically with camera count.

Table 19: OC across camera counts with visual procedure views. Mean additionally averages the camera subsets.

Experiment Config					
$K^\dagger$	$H^\dagger$	Views	N_{cfg}	N_{ep}	
$\{4, 8, 16, 32\}$	$\{1, 2, 4, 7\}$	$V \in \{1, \ldots, 5\}$ One subset per V	$4 \times 5 \times 4$	80k	
Results					
Camera count	1	2	3	4	5	Mean
SR	11.3	15.2	16.0	17.4	9.2	13.8
SSR	19.3	11.7	13.1	12.4	14.2	14.1
SC	42.9	50.9	53.6	53.6	47.3	49.6
RecPo	82.1	91.1	85.7	92.9	83.0	87.0

These results concern the fixed camera subsets shown above rather than an average over alternative subsets of each size.

To examine how visual composition depends on procedure support and task sparsity, we report the skill-context diversity and waypoint-gap marginals in Table 20. For these visual results, the standard deviation across evaluation-seed means is 0.0 percentage points for every reported marginal; this does not measure variation across configurations.

Table 20: OC by skill-context diversity and waypoint gap with visual procedure views. Both marginals average the same five camera subsets.

Experiment Config					
Report	K	H	Views	N_{cfg}	N_{ep}
K	$\{4, 8, 16, 32\}$	$\{1, 2, 4, 7\}^\dagger$	$V \in \{1, \ldots, 5\}^\dagger$ One subset per V	$4 \times 5 \times 4$	80k
H	$\{4, 8, 16, 32\}^\dagger$	$\{1, 2, 4, 7\}$	$V \in \{1, \ldots, 5\}^\dagger$ One subset per V	$4 \times 5 \times 4$	80k

Results								
Method	Skill-context diversity K				Waypoint gap H			
	4	8	16	32	1	2	4	7
SR	15.9	11.5	14.2	13.7	49.4	0.0	0.0	5.8
SSR	20.8	9.6	15.3	10.9	33.4	7.3	5.8	10.1
SC	43.6	47.9	58.6	48.6	40.0	47.1	48.6	62.9
RecPo	60.0	95.0	92.9	100.0	94.3	91.4	86.4	75.7

Mean OC decreases from 94.3% at $H = 1$ to 75.7% at $H = 7$. Across diversity settings, it ranges from 60.0% to 100.0% without a monotonic increase.

C.2.4 Prototype-based Skill Routing

To test whether compact summaries of skill records improve retrieval, we adapt prototype-based routing [29] to SR and SSR. We construct five KMeans prototypes per skill in each procedure view from record transitions and score each skill by its nearest prototype, averaging distances across views. For SSR, prototype summaries also replace record-level evidence in sequence scoring, while the sequence-search procedure is unchanged. The SR and SSR rows in Table 21 are the main-paper SR and SSR at $V = 8$, while the prototype variants retain all represented skills as candidates.

RecPo denotes the main-paper RecPo with the supplied full-resolution procedure views ($V = 8$). For the task specification, all methods receive the same sparse waypoints. For procedure evidence, all methods use sparse procedure-record endpoints under $V = 8$ views, with the prototype variants replacing record-level evidence by the summaries defined above. The comparison therefore measures prototype routing within each retrieval baseline, while RecPo provides a reference under the same task inputs.

Table 21: OC across waypoint gaps for the main-paper retrieval baselines and their prototype variants at $V = 8$. Mean additionally averages H.

Experiment Config					
$K^\dagger$	H	Views	N_{cfg}	N_{ep}	
$\{4, 8, 16, 32\}$	$\{1, 2, 4, 7\}$	$V = 8$	$4 \times 1 \times 4$	16k	
Results					
Method	$H = 1$	$H = 2$	$H = 4$	$H = 7$	Mean
SR	44.5	4.2	0.0	0.0	12.2
SR+prototype	41.3	5.4	0.0	0.0	11.7
SSR	71.2	47.0	8.5	3.5	32.5
SSR+prototype	98.7	94.4	28.0	0.0	55.3
RecPo	100.0	100.0	100.0	100.0	100.0

Prototypes increase mean SSR OC from 32.5% to 55.3%, while the corresponding SR means are 12.2% and 11.7%. The SSR improvement holds at $H = 1, 2, 4$, but its prototype variant reaches 0.0% at $H = 7$. The benefit therefore depends on the retrieval method and waypoint gap.

To characterize prototype retrieval across view configurations and task sparsity, we evaluate the five-prototype variants over the full sweep in Section B.4. Table 22 reports the resulting waypoint-gap marginals.

Table 22: OC of the main-paper retrieval baselines and their five-prototype variants across waypoint gaps, averaged over skill-context diversity and all supplied view configurations. Mean additionally averages H.

Experiment Config								
$K^\dagger$	H	Views†			N_{cfg}		N_{ep}	
$\{4, 8, 16, 32\}$	$\{1, \ldots, 7\}$	$V \in \{3, 4, 5, 8\}$ 7 configurations total			$4 \times 7 \times 7$		196k	
Results								
Method / H	1	2	3	4	5	6	7	Mean
SR	43.8	6.6	0.0	0.0	3.2	3.5	0.0	8.1
SR+prototype	41.4	6.7	0.0	0.0	10.6	0.6	0.0	8.5
SSR	68.7	46.1	37.6	9.1	6.6	1.3	3.5	24.7
SSR+prototype	98.1	90.3	39.0	27.6	1.5	5.9	0.0	37.5
RecPo	92.3	89.8	89.3	89.8	91.8	95.4	94.9	91.9

Across all configurations, SSR+prototype attains higher mean OC than SR+prototype, with averages of 37.5% and 8.5%, respectively. Both reach 0.0% at $H = 7$, and neither exhibits a monotonic trend across all gaps.

C.3 Dense Task Specifications

To compare RecPo with composition from dense specifications, we evaluate adapted sub-trajectory retrieval and graph-search mechanisms. Direct comparisons with the complete prior systems would change the learning or control interface. STRAP [24] and graph-based imitation methods [34, 35] use retrieved or stitched evidence for policy learning. Goal-conditioned graph planners [65, 66] instead assume a goal-conditioned controller and learned value or temporal-distance representations. Our setting provides a fixed library of executable skills without these learned planning components. We therefore adapt their sub-trajectory retrieval and graph-search mechanisms to select skills from the existing library without policy training.

To compare these mechanisms across procedure-support conditions, we average over the KB settings listed in each condition table. In the sub-trajectory retrieval and graph-search comparisons below, RecPo denotes the reference evaluated with the supplied full-resolution procedure views ($V = 8$) and the task inputs specified below. Methods that use procedure views receive this configuration unless stated otherwise.

C.3.1 Main Evaluation with Dense Task Specifications

Tables 23 and 24 report the main evaluation with dense task specifications. To let SR and SSR exploit the dense trajectory, we additionally equip them with bisection goals. Each goal is the best-scoring midpoint of a four-level recursive bisection of the task frames that follow the frame nearest the current observation.

With these goals, SR and SSR recover most subtasks by following the demonstrated frames, and SR exceeds RecPo in OC. Under the sparse waypoints of Tables 9 and 12, the omitted frames leave retrieval without goals to follow, whereas RecPo attains similar OC under both specifications by composing skills from view-level changes. This contrast locates the contribution of RecPo in sparse governing information, where a task specification gives abstract goals rather than the behavior that realizes them.

Table 23: *Ordered completion* and task-inference time by *skill-context diversity* K with dense task specifications. Overall averages all configurations. Both error terms describe variation across configuration means. *SR and SSR with bisection goals.

Experiment Config				
K	Task input	Views†	N_{cfg}	N_{ep}
$\{4, 8, 16, 32\}$	Dense	$V \in \{3, 4, 5, 8\}$ 7 configurations total	4×7	28k

Results

	Skill Retrieval (SR)*		Skill Sequence Retrieval (SSR)*		Skill Chaining (SC)		RecPo (ours)	
K	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)
4	98.9±0.1	14.5±3.9	68.4±3.8	36.6±7.3	46.9±37.5	7.7±2.6	77.6±23.1	7.9±2.6
8	99.0±0.1	17.7±4.7	56.3±11.0	39.8±9.4	26.5±32.4	12.9±4.0	95.9±10.8	13.1±4.0
16	99.2±0.4	23.1±5.1	90.5±1.0	39.6±8.4	14.3±0.0	23.5±6.3	100.0±0.0	23.4±6.3
32	99.1±0.4	32.4±6.7	93.6±0.8	43.5±9.9	14.3±0.0	43.4±12.5	98.0±5.4	43.4±12.6
Overall	99.0±0.3	21.9±8.5	77.2±16.7	39.9±8.7	25.5±27.0	21.9±15.6	92.9±15.3	22.0±15.5

C.3.2 Sub-trajectory Retrieval under Reduced Task Evidence

To assess how sub-trajectory retrieval depends on task-observation density, we reduce the retained task evidence while keeping the stored procedure trajectories dense. The adaptation matches task segments to stored sub-trajectories using subsequence dynamic time warping (S-DTW) and executes the skills associated with the retrieved evidence. S-DTW adds linear interpolation of a sparse task

Table 24: *Ordered completion* and task-inference time by supplied *view resolution* V with dense task specifications. Both error terms describe variation across configuration means. *SR and SSR with bisection goals.

Experiment Config				
$K^{\dagger}$	Task input	Views	N_{cfg}	N_{ep}
$\{4, 8, 16, 32\}$	Dense	$V = 3$ 1 configuration	4×1	4k
$\{4, 8, 16, 32\}$	Dense	$V = 4$ 2 configurations†	4×2	8k
$\{4, 8, 16, 32\}$	Dense	$V = 5$ 3 configurations†	4×3	12k
$\{4, 8, 16, 32\}$	Dense	$V = 8$ 1 configuration	4×1	4k

Results

	Skill Retrieval (SR)*		Skill Sequence Retrieval (SSR)*		Skill Chaining (SC)		RecPo (ours)	
V	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)	OC↑(%)	Time↓(ms)
3	99.0±0.3	15.5±6.1	75.3±19.8	29.6±1.4	14.3±0.0	15.0±10.6	75.0±24.4	15.2±10.4
4	99.1±0.2	19.1±6.8	76.0±18.5	35.1±2.3	19.6±21.5	18.5±12.6	94.6±15.2	18.4±12.3
5	99.0±0.3	22.7±7.2	76.4±17.7	40.7±2.8	22.6±19.7	22.1±14.6	95.2±11.1	22.3±14.7
8	99.2±0.2	31.8±10.1	84.0±10.1	57.3±4.8	57.1±49.5	34.9±24.6	100.0±0.0	35.0±24.5

query onto its original temporal grid before matching. From one 383-frame task demonstration, we uniformly retain observations while preserving the first and final observations and their timestamps. Both methods receive the same interpolated task record as a dense task specification, from which RecPo selects task observations by its task-query abstraction. For procedure evidence, S-DTW uses dense, unsegmented trajectories, whereas RecPo uses sparse initiation and termination observations organized into procedure records under $V = 8$ views. Table 25 varies the number of retained observation steps.

Table 25: OC as fewer task observations are retained before temporal reconstruction. Task demo specs denote the number of retained observation steps.

Experiment Config				
$K^{\dagger}$	H	Views	N_{cfg}	N_{ep}
$\{4, 8, 16, 32\}$	Dense	RecPo: $V = 8$ S-DTW: –	4×1	4k

Results

Task demo specs	383	192	96	48	24	12	6	3	2
RecPo	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
S-DTW	100.0	85.7	85.7	96.4	92.9	57.1	28.6	0.0	0.0

Both methods achieve 100.0% OC with the complete task record. S-DTW varies nonmonotonically at intermediate retention levels and reaches 0.0% with three or two retained observation steps, whereas RecPo remains at 100.0% throughout this experiment. This result concerns reconstruction from timestamped observations of this demonstration, with method-specific task processing applied afterward.

C.3.3 Graph Search over Stored Trajectories

To assess trajectory-graph composition under sparse task specifications, we vary the waypoint gap while retaining the same dense procedure trajectories. We adapt the offline-state graph and shortest-path search of Test-Time Graph Search [65], a mechanism also used for subgoal selection in Graph-Assisted Stitching [66]. Demonstration states form graph nodes, temporal edges preserve observed order, and nearest-neighbor edges connect states across trajectories. Our adaptation uses squared Euclidean state distances and fixed temporal-step and stitching penalties, then applies Dijkstra's algorithm between supplied waypoints and executes the skills along the resulting path. We set the cross-trajectory edge penalty to 2, twice the temporal-step cost. For the task specification, both methods receive the same sparse waypoints without interpolation at each $H \in \{1, \ldots, 7\}$. For procedure evidence, graph search uses dense, unsegmented trajectories without procedure views, while `RecPo` uses sparse procedure-record endpoints under $V = 8$ views. The comparison in Table 26 therefore differs in procedure evidence and composition under the same task inputs.

Table 26: OC across waypoint gaps for graph search.

Experiment Config							
$K^\dagger$	H		Views		N_{cfg}		N_{ep}
$\{4, 8, 16, 32\}$	$\{1, \ldots, 7\}$		`RecPo`: $V = 8$ Others: –		4×7		28k
Results							
Method / H	1	2	3	4	5	6	7
`RecPo`	100.0	100.0	100.0	100.0	100.0	100.0	100.0
Graph search	46.4	14.3	7.1	0.0	0.0	0.0	0.0

Graph-search OC falls from 46.4% at $H = 1$ to 7.1% at $H = 3$ and 0.0% for $H \geq 4$. `RecPo` remains at 100.0% at every gap.

To isolate the role of procedure support in graph search, we fix $H = 1$ and vary skill-context diversity in Table 27. The graph construction and edge penalty of 2 remain fixed.

Table 27: Graph-search OC across skill-context diversity at $H = 1$.

Experiment Config				
K	H	Views	N_{cfg}	N_{ep}
$\{4, 8, 16, 32\}$	1	–	4×1	4k
Results				
Skill-context diversity K	4	8	16	32
Graph search	100.0	42.9	42.9	0.0

Graph search achieves 100.0% OC at the smallest diversity setting and lower completion at the larger settings. Thus, adding procedure demonstrations does not improve completion in this construction. The aggregate results alone do not identify which graph connections cause the errors.

NeurIPS Paper Checklist

1. **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 contributions.

   Guidelines:

   - The answer [N/A] 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 [N/A] 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.

2. **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.

   Guidelines:

   - The answer [N/A] 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.

3. **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: [N/A]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer [N/A] 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.

4. **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: We provide a clear specification of the framework components, the experimental settings used in our evaluation, and the corresponding evaluation protocol.

   Guidelines:

   - The answer [N/A] 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
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) 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).
     (d) 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.

5. **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: The code, data-generation scripts, and evaluation scripts will be released on GitHub with instructions to reproduce the main experimental results.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.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://neurips.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.

6. **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) necessary to understand the results?

   Answer: [Yes]

   Justification: We specify the experimental settings and implementation details in Section 6 of the main paper and provide additional details in the appendix.

   Guidelines:

   - The answer [N/A] 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.

7. **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.

   Guidelines:

   - The answer [N/A] 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.

8. **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 these details in the experiments section and the appendix.

   Guidelines:

   - The answer [N/A] 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).

9. **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 [N/A] 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).

10. **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 potential impacts in the conclusion section.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] 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).

11. **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., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release high-risk models or datasets such as pre-trained language models, image generators, or Internet-scraped datasets. Our experiments are conducted in a controlled benchmark. Therefore, no additional misuse-specific release safeguards are applicable.

    Guidelines:

    - The answer [N/A] 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.

12. **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: We credit the creators and original owners of the existing assets used in our experiments.

    Guidelines:

    - The answer [N/A] 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.

13. **New assets**

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

    Answer: [Yes]

    Justification: The released code includes documentation for running the experiments and reproducing the reported results.

    Guidelines:

    - The answer [N/A] 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.

14. **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: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] 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.

15. **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: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] 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.

16. **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 rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] 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 in the NeurIPS handbook for what should or should not be described.