ToS: Tree-of-Skill Reinforcement Learning Driven by Behavior Trees for Long-Horizon Manipulation

Yishuai Cai[1,4,*] **Xinglin Chen**[1,4,*] **Yunxin Mao**[1] **Kefei Zhu**[4] **Wenjing Yang**[1]
Yaodong Yang[3,4] **Yuanpei Chen**[2,3,4] **Ji Wang**[1] **Minglong Li**[1,†]
[1]National University of Defense Technology [2]PsiBot [3]Peking University [4]PKU-Psibot Lab

Abstract

Behavior Trees (BTs) are known for their modular nature, making them a promising control architecture for long-horizon manipulation tasks. BT planning leverages high-level symbolic behavior models to generate task-specific BTs. However, low-level skill execution involves dependencies that are difficult to express symbolically, such as precise object poses. Moreover, reactive BT execution involves many-to-many skill handoffs, requiring each skill to terminate in a state compatible with its runtime-selected successor. Skill-chaining methods built around predefined sequences do not directly address such runtime-dependent handoffs. We propose Tree-of-Skill (ToS), a neuro-symbolic framework for dynamic skill chaining in BT-driven reinforcement learning. ToS uses symbolic behavior models to construct a dependency set with optional pruning and to predict the successor at runtime. For pairs in this set, the successor's guiding network generates a handoff goal that conditions the current skill. Goal masks support modular goal exchange in a shared representation, allowing independently trained skills to accept guidance from different successors. Initial training combines goal-conditioned skill learning with supervised guiding-network learning from pairwise rollouts, without supervision of complete BT compositions. Optional task-specific fine-tuning further adapts the skills and guiding networks to a target BT. We evaluate 8 skills on 7 two-step tasks and 5 long-horizon tasks involving 3–5 skill invocations in representative successful executions. ToS generally achieves higher success rates than the evaluated baselines, particularly on unseen BT compositions of trained skill transitions without task-specific fine-tuning. Real-robot experiments further demonstrate its applicability to long-horizon manipulation.

1 Introduction

Behavior Trees (BTs) [14, 23] have attracted significant attention in robotics due to their hierarchical and modular nature: each behavior node, corresponding to a low-level skill, can be independently designed, learned, or tested and then composed into different tree structures to address different tasks. BT planning [9, 10] uses symbolic behavior models [8] to compose skills into task-specific BTs according to given task goals. These models specify preconditions and effects to support high-level reasoning about skill composition.

However, for low-level skill design, there are often many dependencies between skills that are difficult to model or reason about using symbols, such as precise object poses. These unmodeled dependencies may reduce execution performance or even lead to task failure. As shown in Figure 1, the skill `PutIn(C,D)` is explicitly dependent on the skill `Pickup(C)` due to its precondition `Holding(C)`.

[*]Equal contribution. [†]Corresponding author.
Work done while Yishuai Cai and Xinglin Chen were visiting the PKU-Psibot Lab.
{caiyishuai, chenxinglin, liminglong10}@nudt.edu.cn

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

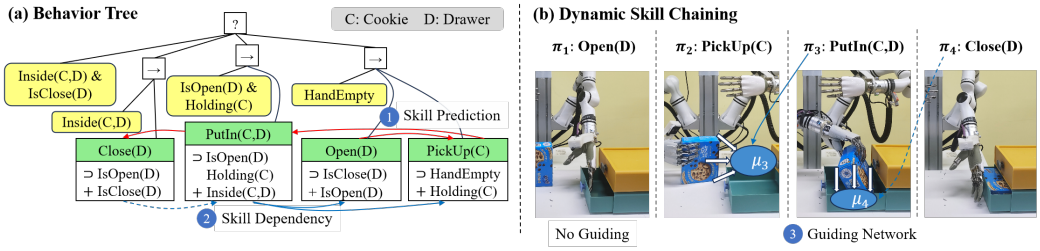


Figure 1: Overview of the ToS framework on a drawer-packing task (C: cookie box, D: drawer). (1) Skill prediction: predicts the runtime successor during BT execution using partial symbolic behavior models ($\supset$: preconditions, $+$: add effects). (2) Dependency check: checks the current skill–predicted successor pair against the dependency set; solid and dashed light-blue arrows denote explicit and implicit dependencies, respectively. (3) Dynamic skill chaining: the predicted successor's guiding network (e.g., μ_3, μ_4) generates a handoff goal to condition the current skill for successor-compatible handoffs.

`Holding(C)` does not fully characterize the initiation region of `PutIn(C,D)`. The preceding skill `Pickup(C)` may satisfy this symbolic condition while terminating in a state from which `PutIn(C,D)` fails. Another example involves `PutIn(C,D)` and `Close(D)`, between which no explicit dependency is modeled. If `Close(D)` is executed after `PutIn(C,D)`, we must ensure the cookie box is laid flat so that the drawer can be closed successfully. This dependency is implicit but crucial for task success.

Prior work studies reinforcement learning (RL) for skill sequencing and chaining [11, 25, 24]. Training around predefined sequences can support runtime policy switching, but switching alone does not specify how a skill's terminal state should accommodate different successors. In a reactive BT, successors are selected from the current state and may impose different continuous-state requirements despite the skill's unchanged symbolic effects. The challenge is to adapt skill termination to the runtime-selected successor while preserving modular training and reuse across BT compositions.

To address this challenge, we propose the Tree-of-Skill (ToS) framework for dynamic skill chaining in BT-driven RL. ToS constructs a conservative dependency set from symbolic behavior models, with optional pruning based on goal masks and domain knowledge. At each control step, it uses these models to predict which successor the BT would select if the current skill succeeded. For skill pairs in the dependency set, the successor's guiding network proposes a handoff goal for the current skill. Goal masks select compatible coordinates in a shared goal representation, while default goals fill unconstrained coordinates. This goal-based interface allows independently trained skills to accept guidance from different successors without conditioning on successor identities.

Training consists of two stages: goal-conditioned skill learning, followed by supervised guiding-network learning from high-return pairwise rollouts. The dependency set specifies the successors used for skill-goal sampling and the pairs used to collect guiding-network training data. These stages require skill- and pair-level data, not supervision of complete BT compositions. Optional task-specific fine-tuning further adapts the invoked skills and guiding networks using rollouts of a target BT.

We evaluate ToS in simulation and on a real robot. The simulation benchmark includes multi-successor tasks and tests composition-level generalization by recombining trained transitions into unseen BTs without task-specific fine-tuning. On the two main unseen compositions, ToS achieves success rates of 52% and 43%, compared with 27% and 15% for the strongest baseline, respectively. These results support reuse of trained transitions rather than generalization to previously untrained skill transitions.

Our main contributions are summarized as follows: (1) We introduce a successor-aware chaining mechanism that combines runtime BT prediction with goal-masked handoff guidance, allowing independently trained skills to accommodate different successors through a shared goal interface. (2) We develop a two-stage training procedure combining goal-conditioned skill learning with supervised handoff-goal learning from pairwise data, without requiring supervision of complete BT compositions. (3) Simulation and real-robot experiments evaluate multi-successor chaining; simulation results further show higher success rates on unseen BT compositions of trained transitions without task-specific fine-tuning.

2 Related Work

BT planning and BT-based RL. BT generation spans learning and evolution [33, 15, 19], search, formal synthesis, and language models [35, 27, 34, 32, 31]. Model-based planners compose behaviors through symbolic preconditions and effects [9, 10, 7], often in STRIPS form [18]; BT-based RL adds behavior learning or online adaptation [22, 26]. ToS instead coordinates continuous handoffs for runtime-selected successors. Online behavior adaptation is therefore complementary to, rather than a direct substitute for, the successor-aware handoff mechanism studied here.

Hierarchical RL and skill chaining. Hierarchical RL provides temporal abstraction [38, 3, 17, 40], while modular policies support cross-task reuse [16, 1, 42]. Skills can be connected through policy sequencing and relay networks [13, 24], or adversarial terminal-state regularization [25]. SeqDex progressively fine-tunes dexterous policies using transition feasibility and supports switching for failure recovery and skipping redundant stages [11]. ToS combines runtime BT selection with a shared, masked goal interface that communicates successor requirements. Pairwise guiding-network training supports reuse of trained transitions in unseen BT compositions without task-specific fine-tuning. ToS emphasizes the handoff interface and pairwise training for composition-level reuse, rather than policy switching alone.

Task and motion planning. TAMP integrates symbolic decisions with continuous feasibility constraints [20], with learned backjumping and dependency-aware heuristics improving search [37, 12]. ToS focuses on learned-skill initiation conditions, which depend on the particular policy as well as geometry. Pairwise execution data yield policy-specific handoff goals for runtime-selected successors, complementing symbolic–continuous planning with execution-level skill coordination.

Programmatic policies. Programmatically interpretable RL [39] learns program-structured policies, whereas Code as Policies [28] and ProgPrompt [36] use language models to generate executable control or task programs. These approaches emphasize policy representation and program generation. ToS addresses a complementary execution-level problem: satisfying the continuous-state requirements of successor skills at handoffs within a reactive BT.

3 Preliminaries

Behavior Tree. A BT is a rooted, directed tree whose control-flow nodes coordinate condition and behavior leaves [14]. In the reactive setting studied here, the root is ticked at each control step to select a behavior from the current state. We use four node types:

- **Condition**: Tests a condition on the current state and returns `success` or `failure`.
- **Behavior**: Executes a skill and returns `success`, `failure`, or `running`. We use *behavior* to distinguish skills from low-level RL actions.
- **Sequence** →: Ticks children from left to right, returning the first `failure` or `running` status, or `success` if all children succeed.
- **Fallback** ?: Ticks children from left to right, returning the first `success` or `running` status, or `failure` if all children fail.

Behavior Model. Let $\mathcal{S}$ be the shared state space, $\mathcal{L}$ a finite set of positive literals, and $\mathcal{C} = 2^{\mathcal{L}}$ the symbolic condition space. Each condition node $\sigma_l : \mathcal{S} \to \{\texttt{success}, \texttt{failure}\}$ evaluates a literal l. The condition $c(s)$ contains the literals that hold in state s; write $c_t = c(s_t)$. A goal condition $c_g \in \mathcal{C}$ holds when $c_g \subseteq c(s)$. For example, `Holding(C)` in Figure 1 indicates that the robot holds the cookie box.

Each behavior b in the skill library $\mathcal{B}$ consists of a policy π_b and a symbolic model $\mathcal{H}_b = \langle pre(b), add(b), del(b) \rangle$. Preconditions, add effects, and delete effects are subsets of $\mathcal{L}$, with $add(b) \cap del(b) = \emptyset$. Given these models and initial and goal conditions $c_0, c_g \in \mathcal{C}$, BT Expansion [9] generates our task BTs. If $pre(b) \subseteq c_t$, the model predicts the condition after successful execution over k control steps as

$$\hat{c}_{t+k} = f_b(c_t) = \big(c_t \cup add(b)\big) \setminus del(b). \tag{1}$$

The prediction $\hat{c}_{t+k}$ need not equal the observed condition $c(s_{t+k})$ and does not specify the continuous terminal state.

Goal-Conditioned Reinforcement Learning. A fixed projection $\varphi : \mathcal{S} \to \mathbb{R}^N$ extracts shared object-centric goal coordinates. In our implementation, the shared representation has $N = 8M$ coordinates for M objects: each contributes a 3D position, a 4D quaternion, and a scalar articulation coordinate, with unused coordinates masked out. All skills share the goal representation $(g, m) \in \mathcal{G} = \mathbb{R}^N \times \{0, 1\}^N$, where $m_i = 1$ marks a guidance-active coordinate; remaining coordinates use default goal values. This common coordinate system supports goal exchange without implying physical feasibility.

Each skill has a goal-conditioned MDP $\mathcal{M}_b = (\mathcal{S}, \mathcal{G}, \mathcal{A}_b, \mathcal{P}_b, \mathcal{R}_b, \gamma_b, \rho_b)$, where $\mathcal{A}_b$ is the low-level action space, $\mathcal{P}_b(ds' \mid s, a)$ the transition kernel, $\mathcal{R}_b$ the reward, γ_b the discount factor, and ρ_b the initial-state distribution. The policy $\pi_b(da \mid s, g, m)$ maximizes

$$\pi_b^* = \arg\max_{\pi_b} \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma_b^t \mathcal{R}_b(s_t, g, m, a_t)\right], \tag{2}$$

where T is the episode length in control steps, ending at termination or the skill's time limit. We sample $s_0 \sim \rho_b$ and an adapted goal–mask pair (g, m) as in Section 5.1; actions and transitions follow $a_t \sim \pi_b(\cdot \mid s_t, g, m)$ and $s_{t+1} \sim \mathcal{P}_b(\cdot \mid s_t, a_t)$.

For a fixed trained policy, the initiation set $\mathcal{I}_b \subseteq \mathcal{S}$ contains states from which the skill meets its success criterion within its execution horizon with probability at least $\eta \in (0, 1)$. Success is evaluated under the default goal and zero guidance mask, as in successor rollouts. Unlike ρ_b, $\mathcal{I}_b$ describes policy-dependent capability, not a distribution. It is conceptual, not an available membership or sampling oracle; training uses prescribed candidate-goal distributions.

4 Tree-of-Skill Framework

ToS coordinates skill handoffs when the BT selects successors at runtime. It first predicts the next behavior under symbolic success of the current one, then passes a successor-specific handoff goal through a shared, masked interface. Algorithm 1 integrates prediction, dependency checking, and goal-conditioned execution.

4.1 Skill Prediction

Intuitively, we ask the BT which behavior it would select if the current behavior had just succeeded. Let $\mathrm{Tick}_{\mathcal{T}}(s; A, D)$ denote a tick that returns the selected behavior, or $\perp$ if none is selected. A condition node in A returns success, one in D returns failure, and all other nodes evaluate the observed state s. Execution uses $b_t = \mathrm{Tick}_{\mathcal{T}}(s_t; \emptyset, \emptyset)$; prediction overrides the current behavior's effects:

$$b'_t = \mathrm{Tick}_{\mathcal{T}}\big(s_t; add(b_t), del(b_t)\big). \tag{3}$$

This implements a one-step symbolic prediction without simulating the future continuous state. The current behavior, predicted successor, and guidance are recomputed at every control step. Re-ticking can respond to observable disturbances, but cannot in general correct a systematically wrong effect model.

4.2 Dependency Check

A skill dependency means that executing b changes quantities that affect the subsequent skill b'. ToS constructs a conservative candidate set algorithmically from the behavior models (Algorithm 4). For each ordered pair $b \neq b'$, it discards the pair if $pre(b') \cap del(b) \neq \emptyset$, since b' cannot immediately follow a successful b under the model. Otherwise, it labels the pair explicit if $pre(b') \cap add(b) \neq \emptyset$ and admits all remaining pairs as potentially implicit dependencies. This inclusive rule captures `PutIn(C,D)`→`Close(D)`: the box pose affects closing even though it establishes no symbolic precondition of Close (Figure 1).

The model-enumerated candidates may be pruned for efficiency. Pairs with $m_b^{in} \odot m_{b'}^{out} = \mathbf{0}$ cannot exchange guidance through the specified interface; domain knowledge may exclude additional candidates. For our eight skills, 56 ordered pairs yield 10 discarded, 9 explicit, and 37 potentially implicit pairs. Of the 31 pruned pairs, 12 have disjoint masks and 19 are removed by manual judgment, leaving 15 retained skill pairs and seven guiding networks. Guidance is applied when $b'_t \in \mathcal{B}_D(b_t)$. The network learns continuous handoff goals for these pairs, not which pairs are dependent.

Algorithm 1 ToS Framework

Input: BT $\mathcal{T}$, initial state s_0, success region $\mathcal{S}_T$, horizon $T_{\max}$
Output: Executed action trajectory

$t \leftarrow 0$
while $s_t \notin \mathcal{S}_T$ and $t < T_{\max}$ **do**
  $b_t \leftarrow \mathrm{Tick}_{\mathcal{T}}(s_t; \emptyset, \emptyset)$ ▷ Select the current skill
  if $b_t = \perp$ **then**
    break
  $b'_t \leftarrow \mathrm{Tick}_{\mathcal{T}}(s_t; add(b_t), del(b_t))$ ▷ Predict under symbolic success
  $g_t \leftarrow g_{b_t}^{def}, \quad m_{g_t} \leftarrow \mathbf{0}$ ▷ Default: no successor guidance
  if $b'_t \neq \perp$ and $b'_t \in \mathcal{B}_D(b_t)$ **then**
    $g_t \leftarrow \mu_{b'_t}(s_t), \quad m_{g_t} \leftarrow m_{b'_t}^{out}$ ▷ Successor guidance for retained pairs
  $m_j \leftarrow m_{b_t}^{in} \odot m_{g_t}$ ▷ Keep mutually supported coordinates
  $\hat{g}_t \leftarrow g_t \odot m_j + g_{b_t}^{def} \odot (1 - m_j)$ ▷ Fill other coordinates with defaults
  $a_t \sim \pi_{b_t}(\cdot \mid s_t, \hat{g}_t, m_j)$
  $s_{t+1} \sim \mathcal{P}_{b_t}(\cdot \mid s_t, a_t)$ ▷ Execute one control step
  $t \leftarrow t + 1$
return $\langle a_0, \ldots, a_{t-1} \rangle$

4.3 Dynamic Skill Chaining

Different successors constrain different coordinates of the same physical state. Building on projected goal representations [2, 43], ToS uses a shared coordinate system and masks to communicate these constraints without changing the current policy's interface.

Goal Mask. Each behavior has fixed input and output masks $m_b^{in}, m_b^{out} \in \{0,1\}^N$, defined on the same object–coordinate indices, and a default goal g_b^{def}. The default is the target for executing the skill in isolation, not a successor-aware handoff. Given (g, m_g), the joint mask $m_j = m_b^{in} \odot m_g$ selects coordinates both supplied by the successor and accepted by the current skill:

$$\hat{g} = g \odot m_j + g_b^{def} \odot (1 - m_j). \tag{4}$$

For PutIn→Close, the intersection covers the box's 7D pose; for Open→PutIn, it covers only the drawer angle (Table 1). This coordinate alignment makes goal exchange well-defined, but does not guarantee physical feasibility.

Guiding Network. The successor's network $\mu_{b'}$ maps the current state to a handoff goal $g_{b'} = \mu_{b'}(s_t)$ intended to steer the predecessor toward $\mathcal{I}_{b'}$. One network is shared across the retained predecessors of b'. It predicts only coordinates selected by $m_{b'}^{out}$; the mask is fixed, not learned. We write $\mu_{b'}(s) \in \mathbb{R}^N$ for the compact output embedded at its shared indices, with zeros elsewhere before Equation 4. For explicit dependencies, output coordinates correspond to objects in the successor's preconditions. For implicit dependencies, domain knowledge informs the output-coordinate selection: Close constrains the box pose although its precondition names only the drawer. Omitting a relevant coordinate prevents communicating it; extra coordinates may reduce sample efficiency. Section 5.2 describes how pairwise rollouts supervise these predictions.

Goal-conditioned Execution. The current skill acts using the adapted goal and joint mask:

$$a_t \sim \pi_{b_t}(\cdot \mid s_t, \hat{g}_t, m_j). \tag{5}$$

Coordinates unused by every dependent successor can be omitted from the policy input. This reduces dimensionality but restricts reuse: a new successor that needs an omitted coordinate may require retraining the predecessor.

5 Tree-of-Skill Training

Training separates goal-reaching capability from handoff selection. We first learn goal-conditioned skills, then freeze them and learn guiding networks from pairwise rollouts. Neither stage requires complete-task supervision; optional fine-tuning uses a target BT.

5.1 Goal-conditioned Skill Learning

ToS supports different skill-learning algorithms. We use SERL [30], following RLPD [6] with Soft Actor-Critic (SAC) [21]. Skill learning is offline-to-online: demonstrations initialize replay buffers, and SAC then learns through environment interaction. Simulation demonstrations are collected through motion planning; real-robot skill learning uses 20 teleoperated demonstrations per skill followed by online training. This stage uses skill-level, not complete-task, supervision.

Goal Sampling. For each dependent successor b', let $q_{b'}$ be a prescribed distribution of candidate handoff goals on the shared coordinates. These distributions use manually specified object-pose and articulation ranges; they need not be supported on $\varphi(\mathcal{I}_{b'})$. At the start of an episode, the training distribution for skill b is

$$(g, m_g) \sim \begin{cases} q_{b'} \otimes \delta_{m_{b'}^{out}}, & \text{w.p. } p_{b'},\ b' \in \mathcal{B}_D(b), \\ \delta_{(g_b^{def}, \mathbf{0})}, & \text{w.p. } p_\emptyset. \end{cases} \tag{6}$$

Here δ denotes a point mass, $\otimes$ pairs a sampled goal with its fixed mask, and $\sum_{b'} p_{b'} + p_\emptyset = 1$. Equation 4 maps this pair to $(\hat{g}, m_j)$ for execution. When $\mathcal{B}_D(b)$ is empty, $p_\emptyset = 1$. Candidate sampling requires no knowledge of the trained successor's exact success set; pairwise rollouts subsequently evaluate which goals yield useful handoffs.

HER. Hindsight Experience Replay (HER) [2] improves sample efficiency by relabeling unsuccessful trajectories with achieved goals.

5.2 Guiding Network Learning

With skill policies frozen, we train one guiding network per successor b' using rollouts from its predecessor set $\mathcal{B}'_D(b') = \{b \in \mathcal{B} : b' \in \mathcal{B}_D(b)\}$.

Candidate evaluation. For each predecessor b, sample $g \sim q_{b'}$ and initialize s_0 to match its output-mask coordinates outside the joint mask (Appendix E.3). Execute π_b with the adapted goal and joint mask; if activated, $\pi_{b'}$ starts from the handoff state s_k under its default goal and zero guidance mask. Here k denotes the observed switch, not a membership test for $\mathcal{I}_{b'}$. Attempts without successor execution count as failed handoffs and are excluded from regression (Appendix A). Each retained two-segment rollout $\tau = (\tau_b, \tau_{b'})$ receives the score

$$G(\tau) = \alpha \sum_{t=0}^{k-1} r_t + (1-\alpha) \sum_{t=k}^{T-1} r_t, \tag{7}$$

where T is the rollout length and $\alpha \in [0, 1]$ balances predecessor goal-reaching and successor execution rewards. A candidate goal does not certify a successful handoff; its utility is assessed through the two executions.

Selection and regression. From I_g attempts per pair, we rank retained rollouts by G and collect up to K predecessor state–goal samples (s_t, g), $t < k$, in descending score order. Samples are pooled across predecessors to minimize the masked MSE in Appendix E.3. Regression targets the sampled goal g, not the achieved state $\varphi(s_k)$. Returns only select data, serving as neither prediction targets nor loss terms. This stage uses no critic, policy-gradient updates, or complete-BT supervision.

5.3 Optional: Task-specific Fine-tuning

The first two stages suffice to compose trained transitions into a new BT. Optional fine-tuning instead adapts the invoked modules to one target task, using full-BT experience rather than isolated skills or pairs.

Fine-tuning Objective. For a task BT $\mathcal{T}$ and initial distribution $\rho_{\mathcal{T}}$, we execute the full tree and adapt the invoked skills and guiding networks to maximize

$$\max_{\{\pi_b, \mu_b\}} \mathbb{E}_{\tau \sim p_{\mathcal{T},\pi,\mu}} \left[\sum_{t=0}^{T-1} \gamma^t \mathcal{R}_{b_t}(s_t, \hat{g}_t, m_{j,t}, a_t) \right]. \tag{8}$$

Here $p_{\mathcal{T},\pi,\mu}$ is the trajectory distribution induced by Algorithm 1, the environment transitions, and $s_0 \sim \rho_{\mathcal{T}}$. The BT selects b_t, and $m_{j,t}$ is the joint mask at control step t.

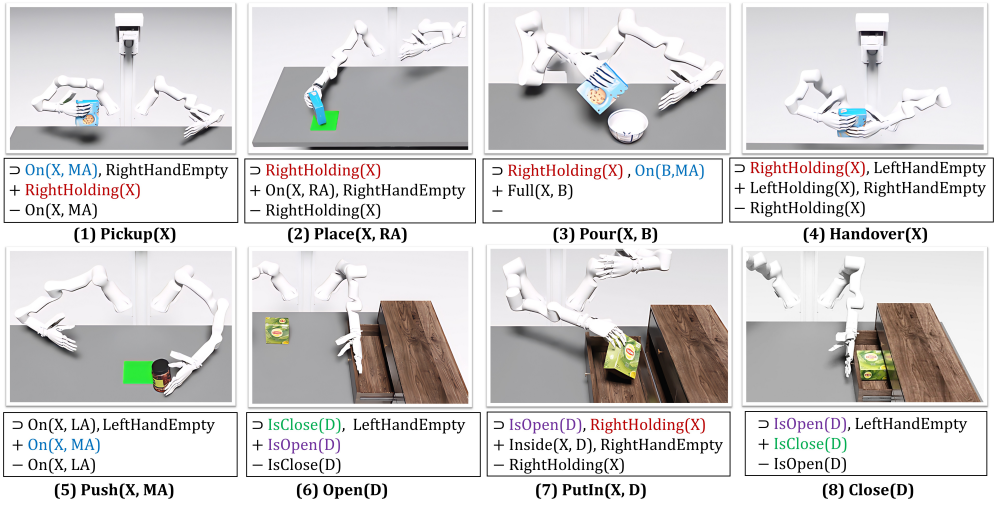


Figure 2: Simulation visualization and behavior models for the 8 skills. Here, *B* denotes the bowl, *D* denotes the drawer; *RA*, *LA*, and *MA* denote the right, left, and middle areas of the table, respectively; *X* represents an object (cookie, green tea, or bowl).

Adaptation Scope. Full-BT rollouts determine which modules are eligible for task-specific updates: at step t, these are the acting skill π_{b_t} and, when guidance is active, the successor's network $\mu_{b'_t}$, not μ_{b_t}. The BT, behavior models, dependency sets, and masks remain fixed. Algorithm 3 gives the task-specific fine-tuning procedure. Each fine-tuned task uses 200K environment steps with a reduced learning rate; Tasks 4 and 5 and the additional unseen compositions receive no task-specific updates (Appendices E.4 and G.2).

6 Experiments

Our experiments evaluate how ToS leverages successor guidance for predecessor execution in multi-step, multi-branch tasks. We train 8 manipulation skills across single- and dual-dexterous-hand settings [4], and construct 7 two-step tasks and 5 long-horizon tasks involving 3–5 skill invocations. We further conduct ablation studies on goal guidance and validate ToS on a real-world robotic platform.

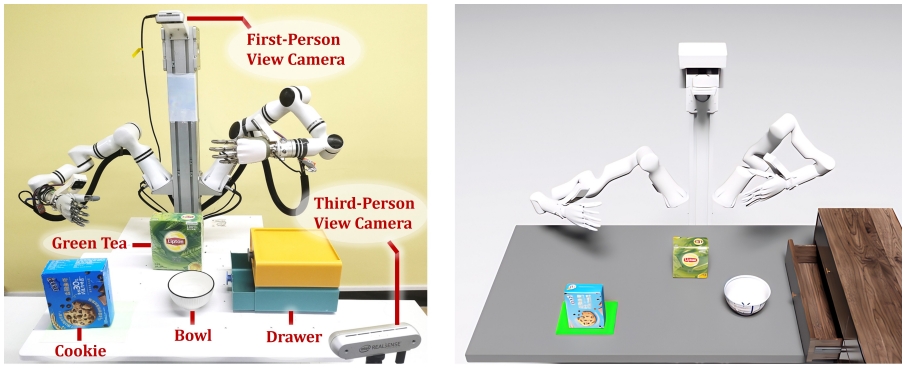
6.1 Experimental Setup

All simulation tasks are built in Isaac Sim on a tabletop with food packages, a bowl, and a pull-out drawer, using a dual-arm robot with two 7-DoF RM75-6F arms and 6-DoF PsiBot G0-R hands. C, T, B, and D denote the cookie box, green tea package, bowl, and drawer; LA, MA, and RA denote the left, middle, and right table areas (real-world perception is described in Section 6.4).

Table 1: Active input and output mask coordinates and single-skill success rates in simulation (%; O denotes the manipulated object). For an object X, P_X and x_X denote its 7D pose and 3D position, and θ_D denotes the drawer's scalar articulation coordinate. All coordinate entries index the shared projection φ; a sum denotes coordinates from both objects.

Skill	Pickup	Place	Pour	Handover	Push	Open	PutIn	Close
Input mask coordinates	P_O	P_O	P_O	P_O	x_O	θ_D	P_C	θ_D
Output mask coordinates	P_O	x_O	$P_O + x_B$	P_O	x_O	P_D	$P_C + \theta_D$	P_C
SR (%)	$96.0_{\pm 4.2}$	$88.0_{\pm 2.7}$	$90.0_{\pm 5.0}$	$79.0_{\pm 6.5}$	$87.0_{\pm 4.5}$	$76.0_{\pm 6.5}$	$81.0_{\pm 6.5}$	$92.0_{\pm 2.7}$

Skills and evaluation protocol. Figure 2 illustrates the eight skills: Pickup, Place, Pour, Handover, Push, Open, PutIn, and Close. Observations include object poses and articulation, robot proprioception, and action history. Table 1 specifies the active goal-mask coordinates and the single-skill



(a) Real (b) Simulation

Figure 3: Comparison between the real and simulated environments used in our experiments.

success rates; Appendix E provides the training configuration. Simulation results report means with one seed-level sample SD shown as a subscript across five training seeds (20 evaluation episodes per seed), both rounded to one decimal place. A task step denotes one skill invocation, not one control timestep; each invocation is capped at 300 control steps.

6.2 Long-Horizon Comparison

Baselines. SeqDex [11] chains skills along predefined sequences using backward fine-tuning. Goal-Conditioned RL (GCRL) [29] uses a monolithic policy; we augment it with supervised goal guidance collected using our procedure. Our Modular RL (MRL) baseline, inspired by [1], composes independently learned skills without successor guidance. Together, these baselines distinguish static chaining, goal guidance, and modular reuse. BT-based online behavior adaptation [22] addresses a different problem from successor-aware termination; we discuss this distinction in Section 2. Appendix G.2 summarizes training access for the compared methods.

Tasks and generalization. Table 2 specifies five tasks through symbolic conditions and example executions. $\mathrm{On}(X, A)$ denotes object X in area A; $\mathrm{Full}(T, B)$, completed pouring from T into bowl B; $\mathrm{Inside}(C, D)$, C inside D; $\mathrm{IsClose}(D)$, a closed drawer; and $\mathrm{LeftHolding}(T)$, T held by the left hand. Initial conditions summarize predicates, not the full state distribution. Appendix B provides task illustrations and descriptions.

Tasks 2, 3, and 5 contain transitions to multiple possible successors, as marked by Branching. Tasks 1–3 are available for task-level training, whereas Tasks 4 and 5 are unseen BT compositions: neither their full BTs nor their task-level rollouts are used for fine-tuning. Their required guided transitions have already appeared in pairwise training, and their objects and initial-state distributions belong to the same families as training. This evaluates composition-level generalization, not generalization to unseen skill transitions or initial-state distributions. All task BTs are generated by BT Expansion [9].

Results. Table 2 reports mean success rates of 73.0% for ToS and 74.0% for SeqDex on Task 1, which is neither branching nor unseen. ToS achieves higher mean success rates than the baselines on Tasks 2–5. On the seen branching Tasks 2 and 3, it improves over SeqDex by 15.0 and 20.0 percentage points, respectively; on the non-branching unseen Task 4, the improvement is 25.0 percentage points. This pattern supports the value of dynamic chaining and modular reuse, although cross-method comparisons do not isolate every architectural difference. Fine-tuning improves the seen tasks; the unseen-task results are obtained without it. Four additional unseen compositions reported in Appendix G.1 further evaluate reuse of trained transitions.

6.3 Ablations

We evaluate seven two-step tasks: six with explicit dependencies and PutIn→Close with an implicit dependency (Figure 2). Table 3 compares default goals, random successor-region goals, learned guidance without fine-tuning, and full ToS.

Table 2: Task specifications and long-horizon success rates (%) in simulation.

(a) Task specifications

Task	Branching	Initial conditions	Target conditions	Example skill execution
Seen tasks	*Available for task-level training*			
Task 1	–	$\mathrm{On}(C, \mathrm{LA})$	$\mathrm{On}(C, \mathrm{RA})$	Push $\rightarrow$ Pickup $\rightarrow$ Place
Task 2	✓	$\mathrm{On}(C, \mathrm{MA})$ $\wedge$ $\mathrm{On}(T, \mathrm{MA})$ $\wedge$ $\mathrm{On}(B, \mathrm{MA})$	$\mathrm{On}(C, \mathrm{RA})$ $\wedge$ $\mathrm{Full}(T, B)$	Pickup $\rightarrow$ Place $\rightarrow$ Pickup $\rightarrow$ Pour
Task 3	✓	$\mathrm{On}(T, \mathrm{LA})$ $\wedge$ $\mathrm{On}(B, \mathrm{LA})$	$\mathrm{Full}(T, B)$ $\wedge$ $\mathrm{On}(T, \mathrm{RA})$	Push $\rightarrow$ Pickup $\rightarrow$ Push $\rightarrow$ Pour $\rightarrow$ Place
Unseen compositions	*No task-specific fine-tuning*			
Task 4	–	$\mathrm{On}(C, \mathrm{MA})$ $\wedge$ $\mathrm{IsClose}(D)$	$\mathrm{Inside}(C, D)$ $\wedge$ $\mathrm{IsClose}(D)$	Open $\rightarrow$ Pickup $\rightarrow$ PutIn $\rightarrow$ Close
Task 5	✓	$\mathrm{On}(C, \mathrm{MA})$ $\wedge$ $\mathrm{On}(T, \mathrm{MA})$ $\wedge$ $\mathrm{IsClose}(D)$	$\mathrm{Inside}(C, D)$ $\wedge$ $\mathrm{LeftHolding}(T)$	Open $\rightarrow$ Pickup $\rightarrow$ PutIn $\rightarrow$ Pickup $\rightarrow$ Handover

Branching marks multiple possible successors (✓: yes, shaded; –: no). $\wedge$ denotes conjunction. Executions illustrate successful runs of 3–5 skill invocations, not prescribed sequences for ToS.

(b) Long-horizon success rates (%)

Method	**Seen tasks**			**Unseen compositions**		
	Task 1	Task 2	Task 3	Task 4	Task 5	**Avg.**
Branching	–	✓	✓	–	✓	–
SeqDex	**74.0**$_{\pm 7.4}$	49.0$_{\pm 4.2}$	42.0$_{\pm 4.5}$	27.0$_{\pm 8.4}$	15.0$_{\pm 7.9}$	41.4
GCRL	65.0$_{\pm 7.1}$	54.0$_{\pm 4.2}$	48.0$_{\pm 5.7}$	17.0$_{\pm 6.7}$	14.0$_{\pm 10.2}$	39.6
MRL	51.0$_{\pm 5.5}$	41.0$_{\pm 4.2}$	37.0$_{\pm 6.7}$	25.0$_{\pm 3.5}$	13.0$_{\pm 8.4}$	33.4
ToS w/o FT	68.0$_{\pm 6.7}$	58.0$_{\pm 4.5}$	53.0$_{\pm 4.5}$	**52.0**$_{\pm 4.5}$	**43.0**$_{\pm 2.7}$	54.8
ToS (ours)	73.0$_{\pm 2.7}$	**64.0**$_{\pm 2.2}$	**62.0**$_{\pm 2.7}$	**52.0**$_{\pm 4.5}$	**43.0**$_{\pm 2.7}$	**58.8**

Avg. equally weights the five tasks; FT denotes fine-tuning. Of the two ToS variants, only ToS (ours) uses FT on Tasks 1–3. Bold entries mark the highest mean in each column, including ties.

Table 3: Ablation of goal guidance on seven two-step tasks in simulation (%). Avg. equally weights the seven tasks. Bold entries mark the highest mean in each column. "Default goals" uses default goals only; "Random goals" samples goals uniformly from the successor's training region; "ToS w/o FT" uses learned guidance without task-specific fine-tuning (FT).

Method	Pickup $\rightarrow$ Place	Pickup $\rightarrow$ Pour	Pickup $\rightarrow$ Handover	Push $\rightarrow$ Pickup	Push $\rightarrow$ Pour	Open $\rightarrow$ PutIn	PutIn $\rightarrow$ Close	Avg.
Default goals	12.0$_{\pm 5.7}$	7.0$_{\pm 5.7}$	6.0$_{\pm 4.2}$	16.0$_{\pm 4.2}$	13.0$_{\pm 5.7}$	7.0$_{\pm 8.4}$	5.0$_{\pm 3.5}$	9.4
Random goals	76.0$_{\pm 5.5}$	71.0$_{\pm 4.2}$	56.0$_{\pm 7.4}$	66.0$_{\pm 6.5}$	61.0$_{\pm 7.4}$	47.0$_{\pm 11}$	59.0$_{\pm 8.2}$	62.3
ToS w/o FT	89.0$_{\pm 5.5}$	82.0$_{\pm 5.7}$	75.0$_{\pm 6.1}$	78.0$_{\pm 5.7}$	78.0$_{\pm 5.7}$	64.0$_{\pm 8.2}$	71.0$_{\pm 7.4}$	76.7
ToS (ours)	**92.0**$_{\pm 2.7}$	**88.0**$_{\pm 5.7}$	**80.0**$_{\pm 7.9}$	**84.0**$_{\pm 4.2}$	**82.0**$_{\pm 5.7}$	**69.0**$_{\pm 6.5}$	**76.0**$_{\pm 4.2}$	**81.6**

Goal guidance. Averaged over the seven tasks, success is 9.4% with default goals, 62.3% with random successor-region goals, and 81.6% with learned guidance and fine-tuning. The default goal is well-defined but successor-blind: completing the current skill alone need not leave a suitable state for the next. Random guidance already improves chaining, and learning further improves goal selection. For the implicit PutIn$\rightarrow$Close transition, default/random/full-ToS success rates are 5.0%, 59.0%, and 76.0%; for the explicit Open$\rightarrow$PutIn transition, they are 7.0%, 47.0%, and 69.0%.

Fine-tuning. Removing fine-tuning produces a modest drop on the two-step tasks, indicating that pairwise guidance is useful before task-specific adaptation.

6.4 Real-World Evaluation

Real-robot evaluation uses the dual-arm platform and demonstration-plus-online training of Section 5.1, not zero-shot sim-to-real policy transfer. An egocentric RealSense D455 supplies object poses through FoundationPose [41]; a lateral camera estimates the drawer angle.

Table 4: Simulation and real-robot success rates (%) for skills and task compositions.

(a) Single skills
20 real trials per skill

Skill	Sim	Real	
	(%)	(%)	k/n
Pickup	96.0	85.0	[17/20]
Place	88.0	80.0	[16/20]
Pour	90.0	75.0	[15/20]
Handover	79.0	65.0	[13/20]
Push	87.0	85.0	[17/20]
Open	76.0	70.0	[14/20]
PutIn	81.0	70.0	[14/20]
Close	92.0	90.0	[18/20]
Mean	**86.1**	**77.5**	**[124/160]**

Reporting (both panels). Sim: means from Table 1 for (a), and from Tables 2 and 3 for (b). Real: success rate with successful trials k out of n in brackets.

Summary. Within each level, Sim is the equally weighted mean; Real is pooled from counts, not rounded percentages. Skill and task evaluations are separate batches.

(b) Skill compositions
24 real trials per task

Task	Sim	Real	
	(%)	(%)	k/n
Two-step tasks			
Pickup → Place	92.0	83.3	[20/24]
Pickup → Pour	88.0	66.7	[16/24]
Pickup → Handover	80.0	54.2	[13/24]
Push → Pickup	84.0	79.2	[19/24]
Push → Pour	82.0	62.5	[15/24]
Open → PutIn	69.0	62.5	[15/24]
PutIn → Close	76.0	58.3	[14/24]
Mean	**81.6**	**66.7**	**[112/168]**
Long-horizon tasks			
Task 1	73.0	58.3	[14/24]
Task 2	64.0	54.2	[13/24]
Task 3	62.0	50.0	[12/24]
Task 4	52.0	41.7	[10/24]
Task 5	43.0	25.0	[6/24]
Mean	**58.8**	**45.8**	**[55/120]**

Table 4 reports single-skill means of 86.1% in simulation and 77.5% on the robot (an 8.6-point decrease). Pooled real success is 66.7% for two-step and 45.8% for long-horizon tasks (14.9- and 13.0-point decreases). Observed difficulties include lighting-sensitive pose estimation and differences in contact friction, arm control frequency, and achievable per-step motion. PID tuning mitigates but does not eliminate them; aggregate gaps do not isolate their causal contributions.

7 Limitations

ToS uses a predefined skill library, symbolic behavior models, and a compact goal representation. A conservative dependency set is constructed automatically from the symbolic models; our experiments use domain-informed pruning and manually specified policy-sensitive mask coordinates. Scaling the library increases pairwise training costs and may require additional guiding networks. Re-ticking updates skill selection but does not repair inaccurate symbolic effects. Evaluation covers tabletop manipulation and recombination of trained transitions in unseen BT compositions; new guided pairs require additional stage-two training.

8 Conclusion

We presented ToS for dynamic skill chaining in BT-driven RL. Symbolic successor prediction, learned handoff goals, and shared-coordinate goal masks allow independently trained skills to be recomposed at execution time. Simulation and real-robot evaluations support the usefulness of this design, including generalization to unseen BT compositions of trained transitions. Future work will reduce reliance on manually specified models and goal representations while extending the range of reusable skills.

Acknowledgments

This work was supported by the National Science Fund for Distinguished Young Scholars (Grant No. 62525213), the National Natural Science Foundation of China (Grant Nos. 62572480 and 62676423), the Furong Plan Young Talents Program of Hunan Province (Grant No. 2026RC3114), and the University Youth Independent Innovation Science Foundation (Grant No. ZK25-11).

References

[1] Jacob Andreas, Dan Klein, and Sergey Levine. Modular Multitask Reinforcement Learning with Policy Sketches. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 166–175. PMLR, August 2017.

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[3] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[4] Fengshuo Bai, Yu Li, Jie Chu, Tawei Chou, Runchuan Zhu, Ying Wen, Yaodong Yang, and Yuanpei Chen. RetrDex: Efficient object retrieval in cluttered scenes with a dexterous hand. In *2026 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2026. arXiv:2502.18423.

[5] Fengshuo Bai, Yufeng Li, Ruihai Wu, Peishuo Wang, Yuhan Wang, Bernie Hao Zhu, Yuanfei Wang, Tawei Chou, Jing Gao, Runchuan Zhu, Ying Wen, Yaodong Yang, and Yuanpei Chen. SafeLab: An interactive high-fidelity benchmark for embodied safety in scientific robotics. In *Proceedings of the Forty-third International Conference on Machine Learning (ICML)*, 2026.

[6] Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. *arXiv preprint arXiv:2302.02948*, 2023.

[7] Yishuai Cai, Xinglin Chen, Zhongxuan Cai, Yunxin Mao, Minglong Li, Wenjing Yang, and Ji Wang. MRBTP: Efficient Multi-Robot Behavior Tree Planning and Collaboration. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 39, pages 14548–14557, 2025.

[8] Yishuai Cai, Xinglin Chen, Yunxin Mao, Kun Hu, and Minglong Li. CABTO: Context-aware behavior tree grounding for robot manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 36180–36188. AAAI Press, 2026.

[9] Zhongxuan Cai, Minglong Li, Wanrong Huang, and Wenjing Yang. BT Expansion: A Sound and Complete Algorithm for Behavior Planning of Intelligent Robots with Behavior Trees. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 6058–6065. AAAI Press, 2021.

[10] Xinglin Chen, Yishuai Cai, Yunxin Mao, Minglong Li, Wenjing Yang, Weixia Xu, and Ji Wang. Integrating Intent Understanding and Optimal Behavior Planning for Behavior Tree Generation from Human Instructions. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2024.

[11] Yuanpei Chen, Chen Wang, Fei-Fei Li, and C. Karen Liu. Sequential dexterity: Chaining dexterous policies for long-horizon manipulation. *arXiv preprint arXiv:2309.00987*, 2023.

[12] Bartłomiej Cieślar, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Jorge Mendez-Mendez. Learning long-horizon action dependencies in sampling-based bilevel planning. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pages 4235–4252. PMLR, 2025.

[13] Alexander Clegg, Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Learning to dress: Synthesizing human dressing motion via deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 37(6):1–10, December 2018.

[14] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. CRC Press, 2018.

[15] Michele Colledanchise, Ramviyas Parasuraman, and Petter Ögren. Learning of Behavior Trees for Autonomous Agents. *IEEE Transactions on Games*, 11(2):183–189, 2019.

[16] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *International Conference on Robotics and Automation (ICRA)*, pages 2169–2176. IEEE, 2017.

[17] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13(1):227–303, November 2000.

[18] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, December 1971.

[19] Kevin French, Shiyu Wu, Tianyang Pan, Zheming Zhou, and Odest Chadwicke Jenkins. Learning behavior trees from demonstration. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7791–7797. IEEE, 2019.

[20] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:265–293, 2021.

[21] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 2018.

[22] Marco Iannotta, Johannes A Stork, Erik Schaffernicht, and Todor Stoyanov. On the Fly Adaptation of Behavior Tree-Based Policies through Reinforcement Learning. *arXiv preprint arXiv:2503.06309*, 2025.

[23] Matteo Iovino, Edvards Scukins, Jonathan Styrud, Petter Ögren, and Christian Smith. A survey of Behavior Trees in robotics and AI. *Robotics and Autonomous Systems*, 154, 2022.

[24] Visak CV Kumar, Sehoon Ha, and C Karen Liu. Expanding motor skills using relay networks. In *Annual Conference on Robot Learning (CoRL)*, pages 744–756. PMLR, 2018.

[25] Youngwoon Lee, Joseph J Lim, Anima Anandkumar, and Yuke Zhu. Adversarial skill chaining for long-horizon robot manipulation via terminal state regularization. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 406–416. PMLR, 2022.

[26] Lei Li, Lei Wang, Yuanzhi Li, and Jie Sheng. Mixed Deep Reinforcement Learning-behavior Tree for Intelligent Agents Design. In *ICAART (1)*, pages 113–124, 2021.

[27] Shen Li, Daehyung Park, Yoonchang Sung, Julie A Shah, and Nicholas Roy. Reactive task and motion planning under temporal logic specifications. In *International Conference on Robotics and Automation (ICRA)*, pages 12618–12624. IEEE, 2021.

[28] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[29] Minghuan Liu, Menghui Zhu, and Weinan Zhang. Goal-Conditioned Reinforcement Learning: Problems and Solutions. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 5502–5511, July 2022.

[30] Jianlan Luo, Zheyuan Hu, Charles Xu, You Liang Tan, Jacob Berg, Archit Sharma, Stefan Schaal, Chelsea Finn, Abhishek Gupta, and Sergey Levine. SERL: A software suite for sample-efficient robotic reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16961–16969. IEEE, 2024.

[31] Artem Lykov, Maria Dronova, Nikolay Naglov, Mikhail Litvinov, Sergei Satsevich, Artem Bazhenov, Vladimir Berman, Aleksei Shcherbak, and Dzmitry Tsetserukou. LLM-MARS: Large Language Model for Behavior Tree Generation and NLP-enhanced Dialogue in Multi-Agent Robot Systems. *arXiv preprint arXiv:2312.09348*, 2023.

[32] Artem Lykov and Dzmitry Tsetserukou. LLM-BRAIn: AI-driven Fast Generation of Robot Behaviour Tree based on Large Language Model. *arXiv preprint arXiv:2305.19352*, 2023.

[33] Aadesh Neupane and Michael Goodrich. Learning Swarm Behaviors using Grammatical Evolution and Behavior Trees. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 513–520. IJCAI Organization, 2019.

[34] Aadesh Neupane, Eric G. Mercer, and Michael A. Goodrich. Designing Behavior Trees from Goal-Oriented LTLf Formulas. *arXiv preprint arXiv:2307.06399*, 2023.

[35] Emily Scheide, Graeme Best, and Geoffrey A. Hollinger. Behavior Tree Learning for Robotic Task Planning through Monte Carlo DAG Search over a Formal Grammar. In *International Conference on Robotics and Automation (ICRA)*, pages 4837–4843, Xi'an, China, May 2021. IEEE.

[36] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023.

[37] Yoonchang Sung, Zizhao Wang, and Peter Stone. Learning to correct mistakes: Backjumping in long-horizon task and motion planning. In *Proceedings of the 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 2115–2124. PMLR, 2023.

[38] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999.

[39] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5045–5054. PMLR, 2018.

[40] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 3540–3549. PMLR, 2017.

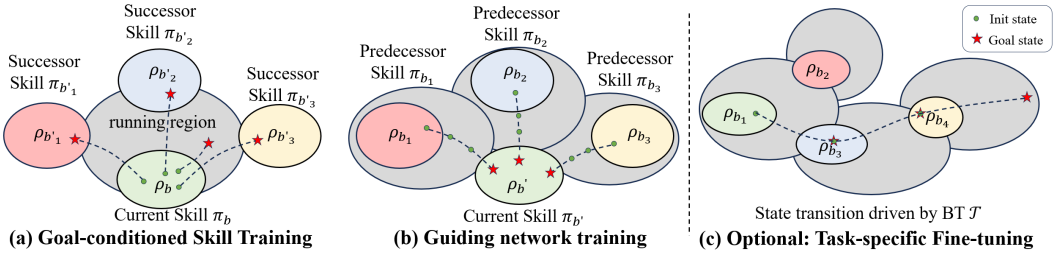
[41] Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. FoundationPose: Unified 6D pose estimation and tracking of novel objects. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17868–17879, 2024.

[42] Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-Task Reinforcement Learning with Soft Modularization. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 4767–4777. Curran Associates, Inc., 2020.

[43] Tianren Zhang, Shangqi Guo, Tian Tan, Xiaolin Hu, and Feng Chen. Generating adjacency-constrained subgoals in hierarchical reinforcement learning. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, volume 33, pages 21579–21590, 2020.

A Details of ToS Training Algorithms

Figure 4 illustrates the two training stages and optional task-specific fine-tuning. Algorithms 2 and 3 give skill and guiding-network training and optional task-specific fine-tuning, respectively. Demonstrations initialize skill replay buffers before online interaction; frozen skill pairs then generate the state–goal data for regression. Candidate goals follow prescribed distributions q_b, not an oracle for the policy-dependent initiation set $\mathcal{I}_b$. The figure is schematic: its ρ_b labels indicate initial-state regions, whereas ρ_b in the text denotes an initial-state distribution, distinct from $\mathcal{I}_b$.



(a) Goal-conditioned Skill Training **(b) Guiding network training** **(c) Optional: Task-specific Fine-tuning**

Figure 4: Overview of ToS training. (a) Goal-conditioned skills learn to reach goals associated with their dependent successors. (b) A successor's guiding network learns handoff goals from predecessor–successor rollouts. (c) Optional task-specific fine-tuning adapts skills and guiding networks using full-BT execution.

Algorithm 2 ToS Training

Input: Behaviors $\mathcal{B}$, MDPs, dependency sets, candidate distributions $\{q_b\}$, mixture weights, α, budgets I_g, K

Output: Trained skills $\{\pi_b\}$ and guiding networks $\{\mu_b\}$, or an insufficient-data report

for each $b \in \mathcal{B}$ **do**
  Initialize the replay buffer $\mathcal{D}_{\pi_b}$ with offline data ▷ Appendix E.1
  while π_b has not converged **do**
    Sample (g, m_g) and adapt it to $(\hat{g}, m_j)$ ▷ Eqs. 6, 4
    Roll out π_b with $(\hat{g}, m_j)$ to obtain τ
    $\tau^* \leftarrow HER(\tau)$
    $\mathcal{D}_{\pi_b} \leftarrow \mathcal{D}_{\pi_b} \cup \{\tau, \tau^*\}$
    Update π_b using $\mathcal{D}_{\pi_b}$ ▷ Eq. 2
for each b' with $\mathcal{B}'_D(b') \neq \emptyset$ **do**
  for each $b \in \mathcal{B}'_D(b')$ **do**
    $\mathcal{J}_{b,b'} \leftarrow \emptyset$
    for $i = 1, 2, \ldots, I_g$ **do**
      Sample $g^i \sim q_{b'}$; initialize s_0 to match its output-mask coordinates outside the joint mask
      Attempt a rollout with frozen π_b under adapted $(g^i, m^{out}_{b'})$; if activated, execute frozen $\pi_{b'}$ under its default goal and zero guidance mask
      if $\pi_{b'}$ was executed **then**
        $\tau^i_{b,b'} \leftarrow (\tau^i_b, \tau^i_{b'}), \quad G^i \leftarrow G(\tau^i_{b,b'})$ ▷ Eq. 7
        $\mathcal{J}_{b,b'} \leftarrow \mathcal{J}_{b,b'} \cup \{(\tau^i_{b,b'}, g^i, G^i)\}$
    $\mathcal{D}_{b,b'} \leftarrow \emptyset$
    while $|\mathcal{D}_{b,b'}| < K$ and $\mathcal{J}_{b,b'} \neq \emptyset$ **do**
      $((\tau^j_b, \tau^j_{b'}), g^j, G^j) \leftarrow PopMaxReturn(\mathcal{J}_{b,b'})$
      Append predecessor pairs (s_t, g^j) to $\mathcal{D}_{b,b'}$, up to K samples
  $\mathcal{D}_{\mu_{b'}} \leftarrow \bigcup_{b \in \mathcal{B}'_D(b')} \mathcal{D}_{b,b'}$
  if $\mathcal{D}_{\mu_{b'}} = \emptyset$ **then**
    return insufficient training data for b'
  Train $\mu_{b'}$ on $\mathcal{D}_{\mu_{b'}}$ by MSE ▷ Eq. 9
return $\{\pi_b\}, \{\mu_b\}$

Handoffs and pair-data eligibility. During BT execution, the next tick determines the active skill from the tree structure, node statuses, and observed conditions. Predecessor failure or timeout does

not force execution of the predicted successor. Handoff goals aim to place the state within the successor's successful execution region; this aim is distinct from an exact test of membership in $\mathcal{I}_{b'}$. An attempt in which the designated successor is never executed counts as a failed handoff. It is excluded from the two-segment score and regression sample selection because no successor segment is available; this exclusion does not change its failure status. Thus, I_g counts rollout attempts, whereas the scored set may be smaller. Eligibility requires successor execution, not successor success.

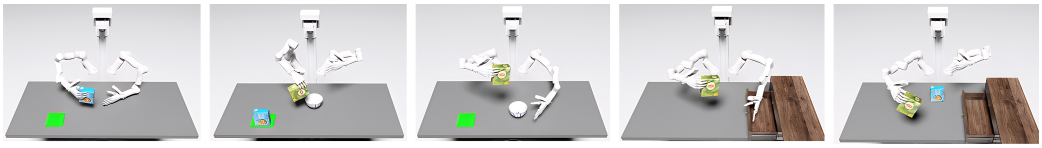
Algorithm 3 Task-specific Fine-tuning for ToS (Optional)

Input: Trained skills $\{\pi_b\}$ and guiding networks $\{\mu_b\}$, MDPs $\{\mathcal{M}_b\}$, BT $\mathcal{T}$, initial distribution $\rho_{\mathcal{T}}$, success region $\mathcal{S}_T$, rollout horizon $T_{\max}$, iterations I_f
Output: Fine-tuned skills $\{\pi_b^*\}$ and guiding networks $\{\mu_b^*\}$

for $i = 1, 2, \ldots, I_f$ **do**
  $s_0 \sim \rho_{\mathcal{T}}$, $t \leftarrow 0$, $\mathcal{B}_i^{\pi} \leftarrow \emptyset$, $\mathcal{B}_i^{\mu} \leftarrow \emptyset$
  while $s_t \notin \mathcal{S}_T$ and $t < T_{\max}$ **do**
    Select $b_t, b_t', \hat{g}_t, m_{j,t}$ and execute a_t as in Algorithm 1; stop if $b_t = \perp$
    Store $(s_t, b_t, b_t', \hat{g}_t, m_{j,t}, a_t, r_t, s_{t+1})$ in task-rollout data
    Add the transition to $\mathcal{D}_{\pi_{b_t}}$; $\mathcal{B}_i^{\pi} \leftarrow \mathcal{B}_i^{\pi} \cup \{b_t\}$
    if $b_t' \neq \perp$ and $b_t' \in \mathcal{B}_D(b_t)$ **then**
      Associate the transition with guiding network $\mu_{b_t'}$; $\mathcal{B}_i^{\mu} \leftarrow \mathcal{B}_i^{\mu} \cup \{b_t'\}$
    $t \leftarrow t + 1$
  Update $\{\pi_b : b \in \mathcal{B}_i^{\pi}\}$ and $\{\mu_b : b \in \mathcal{B}_i^{\mu}\}$ using task-rollout data ▷ Eq. 8
return $\{\pi_b^*\}, \{\mu_b^*\}$

B Long-Horizon Task Descriptions

Figure 5 illustrates the five tasks in Table 2. Their BTs are generated using BT Expansion [9]. The executions below are representative sequences of skill invocations, not control timesteps or prescribed plans. LA, MA, and RA denote the left, middle, and right table areas. Each task requires all its target predicates. Tasks 2, 3, and 5 involve multiple possible successors; Tasks 4 and 5 are unseen BT compositions of trained transitions, with objects and initial states drawn from the training families.



(1) Task1 (2) Task2 (3) Task3 (4) Task4 (5) Task5

Figure 5: Five long-horizon robotic manipulation tasks in the simulation environment.

- **Task 1:** Move the cookie box from LA to RA. A representative successful execution is Push, Pickup, Place.
- **Task 2:** Starting with the cookie box, green tea package, and bowl in MA, move the cookie box to RA and perform the pouring operation from the package into the bowl. A representative successful execution is Pickup, Place, Pickup, Pour.
- **Task 3:** Starting with the green tea package and bowl in LA, perform the pouring operation into the bowl and place the package in RA. A representative successful execution is Push, Pickup, Push, Pour, Place.
- **Task 4:** Starting with the cookie box in MA and the drawer closed, put the box inside and close the drawer. A representative successful execution is Open, Pickup, PutIn, Close.
- **Task 5:** Starting with the cookie box and green tea package in MA and the drawer closed, put the box inside the drawer and transfer the green tea package to the left hand. A representative successful execution is Open, Pickup, PutIn, Pickup, Handover.

C Observation and Action Spaces

C.1 Observation Space

Policy observations contain robot state, action history, object geometry, and goal conditioning. Table 5 gives the base state block for one manipulated object and the active arm. The three hand-state entries

describe opening/closing and thumb rotation; the separate nine-dimensional block stores the most recent input action.

Table 5: Base observation block for a single manipulated object.

Index	Dim.	State component
0–5	6	End-effector position and mixed-angle orientation
6–8	3	Hand opening/closing and thumb-rotation state
9–17	9	Most recent input action
18–20	3	Object position
21–26	6	Object orientation in a 6D rotation representation
27–32	6	Object axis-aligned bounding box
Total	33	

Note. Indices are zero-based. Goal, mask, and skill-specific phase information are not included in this base block.

Goal and mask coordinates are appended according to the skill's interface. The shared representation allocates eight coordinates per object: position (3D), quaternion (4D), and articulation (1D). This is a coordinate convention, not a universal policy-input dimension: compact inputs may omit unused articulation coordinates, include a phase indicator, or include multiple objects. The policy's goal conditioning is distinct from the state features used by the guiding network.

C.2 Action Space

Each control action $\mathbf{a}_t \in \mathbb{R}^9$ specifies six end-effector coordinates and three hand-control coordinates for the active arm (Table 6). The orientation uses the controller's mixed-angle parameterization $(\theta_{\mathrm{rot}}, \theta_y, \theta_z)$, comprising wrist rotation and two directional angles; it is distinct from the quaternion goal representation.

Table 6: Action components for the active arm and hand.

Index	Dim.	Control component
0–2	3	End-effector position
3–5	3	Mixed-angle orientation
6	1	Thumb opening/closing
7	1	Finger opening/closing
8	1	Thumb rotation
Total	9	

Note. Indices are zero-based. The controller's mixed-angle orientation is distinct from the quaternion goal representation.

D Skill Setup and Reward Structure

Table 7 summarizes the eight skills. A default goal specifies the objective of an isolated skill; successor guidance replaces only the coordinates selected by the joint mask in Equation 4.

Reward structure. Skill rewards combine task-dependent shaping terms: grasp/contact alignment, object elevation, target-position and orientation alignment, or drawer articulation. Lifting terms apply to object-lifting skills, not as a universal requirement for all eight skills. Open and Close use an articulation target; the scalar articulation coordinate is distinct from the object's 7D pose.

Execution and evaluation. For Handover, the right-hand policy transfers the object while the left hand waits at a predefined pose and closes using a rule-based controller. Pour uses a local pose-based completion criterion, not a measurement of transferred liquid volume. Local skill completion and training rewards are distinct from long-horizon success, which requires the conjunction of target predicates in Table 2. Initialization ranges do not define the policy-dependent initiation set, and neither a lifting threshold nor a goal-distance threshold is a universal failure condition for this skill library.

Table 7: Skill initialization and default objectives.

Skill	Initial configuration	Default objective
Pickup	Right hand near the object	Grasp and lift the object while preserving its orientation.
Place	Object held above the placement area	Move the object onto the target pad at the designated pose.
Pour	Container held near the bowl	Position and tilt the container above the bowl.
Handover	Object held by the right hand near the waiting left hand	Transfer the object to the left hand.
Push	Left hand near an object on the table	Move the object toward the table center.
Open	Drawer near its closed configuration; left hand available	Move the drawer to its open configuration.
PutIn	Object held above the open drawer	Move the object to the target pose inside the drawer.
Close	Drawer near its open configuration; left hand available	Move the drawer to its closed configuration.

Note. These are local training setups, not fixed execution sequences or complete task-success criteria.

D.1 Per-skill Setup and Completion

The following descriptions distinguish local training objectives from the requirements of a subsequent skill. Goal coordinates follow Table 1; a sampled initialization is not a certificate of successful execution.

Pickup. The right hand starts near the object to establish thumb, index-finger, and palm contact. Reward shaping combines grasp alignment, elevation, and target-position and orientation alignment. The default target lies above the object's initial position with unchanged orientation. Local completion is assessed by object elevation and distance to the commanded position; these conditions alone do not establish compatibility with every successor.

Place. The right hand initially holds the object above the placement area. Grasp-related and target-pose terms guide the descent toward the pad. The default objective specifies the pad pose, while local completion concerns the object's placement height and target proximity. Orientation shaping and the completion test need not constrain identical coordinates.

Pour. The right hand holds the container near the receiving bowl. Grasp, elevation, position, and orientation terms guide the container above the bowl and toward a pouring pose. Local completion uses elevation, container–target distance, and orientation error. It does not measure transferred liquid volume. The bowl's position is also relevant to the entry state that Pour requests from its predecessors.

Handover. The right hand holds the object near the predefined waiting pose of the left hand. Reward shaping combines grasp, elevation, and handover-pose alignment. Local checks involve left-hand contact and object elevation. The learned policy controls the right hand; the left hand remains at its waiting pose and uses rule-based closing to complete the transfer.

Push. The left hand starts near an object resting on the table. Hand–object approach and target-alignment terms encourage motion toward the table center while preserving the object's orientation. Local completion is based on target-position proximity. The goal interface accepts position guidance; unlike grasp-and-lift skills, Push does not require lifting the object to enter its training domain.

Open. The drawer starts near its closed configuration with the left hand available. Shaping uses palm approach to the inner side of the drawer panel and progress toward the target articulation. The default objective is the open configuration, and local completion is assessed against the commanded articulation. When followed by PutIn, Open accepts articulation guidance, not the box-pose coordinates that PutIn may also request from other predecessors.

PutIn. The right hand holds the object above the open drawer. Grasp, elevation, and insertion-pose alignment guide the object toward a target position and orientation inside the drawer. Local completion concerns insertion and target-pose tolerances. If Close is the successor, its guiding network μ_{Close} supplies the box-pose target to π_{PutIn} so that insertion leaves room for closing. This differs from μ_{PutIn}, which supplies entry requirements to predecessors of PutIn.

Close. The drawer starts near its open configuration with the left hand available. Shaping uses palm approach to the outer side of the drawer panel and progress toward the closed articulation; local completion is assessed against that target. The Close policy accepts articulation guidance, whereas its guiding network supplies a box-pose goal to predecessors such as PutIn. The box pose influences closing feasibility even though it is absent from Close's symbolic preconditions.

E Training Details

E.1 Offline RL Data Collection

During skill pre-training, we collect simulation demonstration data using rule-based action sequences and inverse kinematics (IK) interfaces. Each simulated skill uses 50 demonstration trajectories. The collection process generates action sequences using IK solvers, executes them in the simulator with domain randomization, records states and actions at each timestep, and checks the demonstration success criteria. These data initialize the replay buffer before online interaction. The real-robot protocol instead uses 20 teleoperated demonstrations per skill, as specified in Section 5.1.

E.2 RL Training Hyperparameters

We use SERL with SAC and offline replay data following RLPD. Table 8 reports the original skill-training configuration. The evaluation trajectories listed here refer to training-time evaluation, distinct from the final five-seed, 20-episode-per-seed protocol in Section 6.

Table 8: Hyperparameters for SAC skill training.

Parameter	**Value**
Network and initialization	
Actor network (MLP layers)	[256, 256]
Critic network (MLP layers)	[256, 256]
Critic ensemble size	10
Critic subsample size	2
Temperature initialization	1.0×10^{-2}
Optimization and replay	
Learning rate (actor)	3.0×10^{-4}
Learning rate (critic)	3.0×10^{-4}
Learning rate (temperature)	3.0×10^{-4}
Discount factor (γ)	0.99
Soft target update rate (τ)	0.005
Target entropy	Auto-computed
Batch size	256
Critic–actor update ratio	8
UTD ratio	1
Training starts	1,000
Maximum training steps	2,000,000
Replay buffer capacity	100,000,000
Collection and evaluation	
Random steps	1,000
Steps per update	30
Log period	10
Evaluation period	2,000
Evaluation trajectories	5
Checkpoint period	10,000
Warmup steps	2,000
Maximum trajectory length	300

Note. Evaluation trajectories refer to training-time evaluation, not the final five-seed evaluation protocol.

E.3 Guiding Network Training

Network Architecture. Each guiding network is an MLP with hidden widths [256, 256, 128], ReLU activations, and dropout 0.1. The output contains the $d_b^{out} = \|m_b^{out}\|_0$ active goal coordinates in Table 1. The fixed mask is attached by the goal interface and is not a network prediction. Thus, neither an eight-dimensional output nor an additional mask head is common to all skills; Pour, for example, constrains ten coordinates across two objects.

Joint goal and state initialization. Candidate goals specify both commanded coordinates and scene configuration. Coordinates in $m_b^{in} \odot m_{b'}^{out}$ condition the predecessor; remaining coordinates in $m_{b'}^{out}$ are realized through compatible scene initialization and maintained during predecessor execution. For Open→PutIn, we sample a box pose and drawer-articulation target. The robot is initialized holding the box at that pose and maintains it while Open attempts to reach the articulation target. PutIn then evaluates the resulting handoff state. The box-pose label therefore corresponds to the initialized configuration, while the articulation label is the command given to Open; the full candidate goal is retained for regression.

Loss Function and Training. For a nonempty selected predecessor-segment dataset $\mathcal{D}_{\mu_b}$ and $d_b^{out} > 0$, offline regression minimizes

$$\mathcal{L}_b = \frac{1}{|\mathcal{D}_{\mu_b}| d_b^{out}} \sum_{(s,g)\in\mathcal{D}_{\mu_b}} \left\| m_b^{out} \odot \big(\mu_b(s) - g\big) \right\|_2^2 . \tag{9}$$

The target g is the sampled handoff goal associated with a selected rollout, not the successor's return or an achieved-state relabeling. Each selected sample supervises all coordinates in the successor's output mask, not just the joint mask used by its predecessor. For Open→PutIn, these include the initialized box pose and commanded drawer articulation. In the compact implementation, the loss is evaluated on these output-mask coordinates, equivalent to the masked expression above. The sample count is distinct from the shared goal dimension N.

Training configuration. Regression uses Adam with learning rate 10^{-3}, batch size 32, and up to 100 epochs with early stopping based on validation loss. A 20% validation split selects the checkpoint with the lowest validation loss. ReduceLROnPlateau reduces the learning rate by a factor of 0.5 after 10 epochs without improvement.

E.4 Task-specific Fine-Tuning

The optional task-specific stage uses 200,000 environment steps per fine-tuned task and follows Algorithm 3. It is applied only to seen tasks, not to Tasks 4 and 5 or the additional unseen compositions. Table 9 lists its training configuration.

Table 9: Hyperparameters for optional task-specific fine-tuning.

Parameter	Value
Environment steps per task	200,000
Learning rate	1.0×10^{-4}
Discount factor (γ)	0.99
Soft target update rate (τ)	0.005
Batch size	256
Replay buffer capacity	100,000,000

Note. The learning rate is one-third of the initial SAC learning rate, 3.0×10^{-4}. The other listed SAC settings remain unchanged.

E.5 Computing Infrastructure

All experiments were performed on a workstation running Ubuntu 22.04.5 LTS, with an Intel Core i9-14900K CPU (24 physical cores, 32 threads, up to 6.0 GHz) and 64 GB RAM. GPU acceleration used an NVIDIA GeForce RTX 4090 with 24 GB memory, NVIDIA driver 560.35.03, and CUDA toolkit 12.6. The software environment included PyTorch v2.4, TensorFlow v2.19, and JAX v0.4.15.

F Dependency Construction and Pruning

Algorithm 4 formalizes the conservative construction described in Section 4.2. The explicit/implicit labels describe symbolic relationships, not a learned classifier of physical dependence. A pair whose

successor precondition intersects the predecessor's delete effects is excluded only as an immediate transition after a successful execution under the model; the rule does not assert independence under arbitrary failures or intervening actions.

Algorithm 4 Dependency Set Construction

Input: Behaviors $\mathcal{B}$, models $\{\mathcal{H}_b\}$, masks, optional pruning predicate P
Output: Dependency sets $\{\mathcal{B}_D(b)\}$ with explicit/implicit labels

for each $b \in \mathcal{B}$ **do**
  $\mathcal{B}_D(b) \leftarrow \emptyset$
  for each $b' \in \mathcal{B} \setminus \{b\}$ **do**
    if $pre(b') \cap del(b) = \emptyset$ **then**
      if $pre(b') \cap add(b) \neq \emptyset$ **then**
        $kind \leftarrow$ `EXPLICIT`
      else
        $kind \leftarrow$ `IMPLICIT`
      if pruning is disabled or $P(b, b')$ retains the pair **then**
        $\mathcal{B}_D(b) \leftarrow \mathcal{B}_D(b) \cup \{b'\}, \quad label(b, b') \leftarrow kind$
return $\{\mathcal{B}_D(b)\}$

The optional predicate first removes pairs with zero joint masks and may then apply domain-specific pruning. A zero joint mask means that this interface cannot transmit a goal constraint between the pair; it does not establish physical independence. Without pruning, all admissible explicit and potentially implicit pairs remain. For eight skills, enumeration considers 56 ordered pairs: 10 are excluded, 9 are explicit, and 37 are potentially implicit. It takes less than 0.1 ms in our setting. Pruning removes 12 pairs with disjoint masks and 19 additional pairs by judgment, leaving 15 dependencies. Seven successors require guiding networks; a network is shared across its retained predecessors rather than instantiated once per pair.

G Additional Evaluation

G.1 Unseen BT Compositions

Table 10 reports success rates for four additional BT compositions without task-specific fine-tuning. All guided transitions were included in pairwise training; the third composition contains six skill invocations. These point estimates evaluate recombination of trained transitions, not generalization to unseen transitions.

Table 10: Additional unseen BT compositions in simulation.

Skill composition	**Success rate (%)**	
	ToS	**SeqDex**
Push, Pickup, Handover	66	41
Push, Pickup, Pour, Place	55	30
Open, Pickup, PutIn, Pickup, PutIn, Close	35	10
Push, Pickup, PutIn, Close	46	19
Mean	51	25

Note. Values are point estimates without task-specific fine-tuning. All guided transitions were included in pairwise training.

A separate same-BT variant with an unseen target object achieves 49% success, compared with 52% on the original Task 4. This is a distinct object variation rather than an additional unseen BT structure, and a single variant does not establish broad object-level generalization.

G.2 Baseline Training Access

Table 11 distinguishes skill-level learning, pairwise supervision, and task-specific adaptation. It describes training access rather than establishing equal total interaction budgets across methods.

Table 11: Training access and adaptation in the long-horizon comparison.

Method	Representation	Training and adaptation
SeqDex	Reusable skills, fixed sequence	Skill training followed by backward sequence-specific fine-tuning
GCRL	Monolithic goal-conditioned policy	Task-level policy training with supervised goal guidance
MRL	Independently trained skills	Skill training, no successor-provided goal guidance
ToS w/o FT	Skills and guiding networks	Skill-level RL and task-agnostic pairwise regression
ToS	Skills and guiding networks	Same stages, then 200K environment steps per fine-tuned task

Note. FT denotes task-specific fine-tuning. Task-level training uses Tasks 1–3; Tasks 4 and 5 receive no task-specific fine-tuning. Training access does not imply equal total interaction budgets.

G.3 Statistical Reporting

For the original simulation comparisons, each of five training seeds contributes a success rate from 20 evaluation episodes. Means and sample standard deviations are computed over these five rates; the 100 episodes are not independent training replicates. Additional composition and object-variation results are point estimates. Real-robot task results are pooled binomial proportions with the trial counts in Table 4, not seed-level means with simulation SDs.

H Broader Impacts

Reusable robotic skills may reduce the engineering effort needed to automate manipulation tasks. However, perception errors, inaccurate behavior models, and policy failures can lead to unintended physical actions. The reported success rates do not constitute a safety guarantee. Deployment outside the controlled experimental setting requires independent safety assessment, supervision, and physical safeguards; operation around untrained users or in safety-critical settings has not been evaluated [5].

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 claims in the abstract and introduction match the core contributions and are supported by the method and results sections.

   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: The paper includes a dedicated discussion of 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 presents an algorithm and empirical evaluation, not new formal theoretical results or proofs.

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: [No]

Justification: We describe the algorithms, evaluation protocol, skill objectives, and network training configuration, but do not provide a complete specification of all skill rewards, termination thresholds, and pairwise data-collection settings. Code and data are not publicly released.

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: [No]

   Justification: Code and data are not publicly released; the paper provides algorithm descriptions and a partial experimental specification.

   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: [No]

   Justification: The paper specifies the main evaluation protocol and network training configuration, but not all pairwise rollout budgets, goal-sampling settings, or baseline training budgets.

   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: The experimental protocol defines seed-level means and standard deviations; Appendix G explains the statistical unit and aggregation. Real-robot tables state the trial counts, and additional composition results are identified as point estimates.

   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: [No]

   Justification: Appendix E reports hardware and software, and Appendix G.2 states the 200K-step task-specific adaptation budget. Per-experiment execution times and complete compute budgets are not reported.

   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: The research adheres to the NeurIPS Code of Ethics, with no ethical concerns related to data usage, human subjects, or potential misuse identified.

   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: Appendix H discusses potential benefits for robotic manipulation and risks from perception, control, and model errors. We do not claim certified safety or demonstrated societal benefits.

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 models or data with high risk for misuse; thus, no special safeguards are necessary.

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: [No]

Justification: The object and scene assets were scanned and created by the authors. Third-party software and models are credited through citations, but their licenses and terms of use are not fully documented in the paper.

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: [N/A]

    Justification: The object and scene assets were scanned by the authors but are not released with this paper; the method, protocol, and training details are documented in the paper and appendices.

    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 study does not involve crowdsourcing or research on 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 study does not involve crowdsourcing or research on 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: LLMs were used only for language polishing and do not impact the core methodology.

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.