Recover, Discover, Plan: Learning Skills and Concepts from Robot Failures

Bowen Li[1,5*], **Mayank Mishra**[1], **Y. Isabel Liu**[2], **Stone Tao**[3], **Nishanth Kumar**[4], **Alexander Gray**[5], **Ruwan Wickramarachchi**[6], **Jonathan Francis**[1,6], **Sebastian Scherer**[1], **Tom Silver**[2]
[1]CMU, [2]Princeton, [3]AI2, [4]MIT, [5]Centaur AI, [6]Bosch Center for AI

Abstract: Intelligent robots should not only recover from failures, but also acquire the abstract knowledge needed to avoid them in the future. While reinforcement learning (RL) can learn reactive recovery behaviors, training a separate policy for every distinct failure mode is highly inefficient. We introduce Recovery-Driven Synthesis of Relational Concepts (ReSYNC), the first approach that progressively discovers and refines state abstractions (relational predicates) from failure-recovery experience to support abstract planning. Unlike purely reactive methods, ReSYNC jointly learns skills and concepts through an incremental dual-learning process. In the skill-learning phase, the robot uses RL to learn to recover from failures seen in training tasks. In the concept-learning phase, the robot discovers new relational predicates and refines its abstract planning model to explain and generalize the learned recovery behaviors. This interaction enables ReSYNC to convert local recoveries seen during training into global failure avoidance at test time. Across four simulated domains, we show that ReSYNC's ability to continually expand and refine its abstraction library allows it to solve long-horizon, previously unseen problems, outperforming strong baselines by over 50%. Additionally, we demonstrate sim-to-real transfer of ReSYNC, where it performs real-world non-prehensile manipulation skills and generalizes to unseen scenarios through abstract planning. Overall, ReSYNC represents a significant step toward robots that autonomously acquire abstractions for scalable, failure-aware planning in the physical world. Project website: https://jaraxxus-me.github.io/ReSYNC/.

Keywords: Recovery Learning, Skill and Concept Learning, Abstract Planning

1 Introduction

Failure is inevitable for a robot in a *big world* [1]. A rover may slip on ice or get trapped in mud. A mobile manipulator may struggle to retrieve an object from a tight drawer. Some failures can be anticipated and avoided by design, but truly intelligent robots should handle failures on their own.

Previous work has proposed that robots should learn *recovery skills* to escape from failures [2–6]. Recovery skills perform best when some failure modes are repeatedly encountered. But when the world is *big* and novel failure configurations are combinatorial, recovery skill learning alone scales poorly. It treats failures as isolated events rather than exploiting their shared structure.

We propose that robots should treat failures not as impediments necessitating recovery, but rather, as opportunities for *discovery*. We consider a robot with (1) a general library of skills and concepts that are grounded in the physical world; (2) a symbolic planner that composes the skills and concepts given a task; and (3) a curriculum of training tasks designed to elicit failures. When the robot encounters a failure, it initiates a skill-concept dual learning process that augments the skills and concepts in its library. As illustrated in Figure 1, the robot augments its skill set by learning a new recovery skill

*Correspondence to: bowenli2@andrew.cmu.edu, basti@andrew.cmu.com, and tsilver@princeton.edu, work partly done when Bowen Li was an intern at Centaur AI.

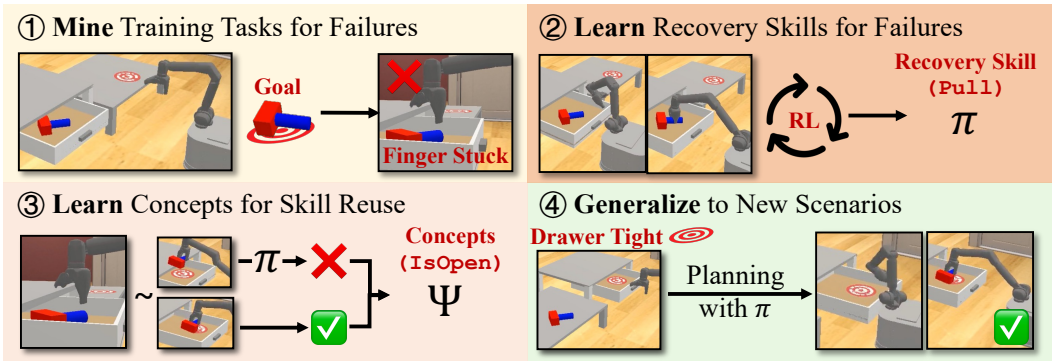

Figure 1: Overview of ReSYNC. The learning in ReSYNC is driven by failures in training tasks, where it starts by acquiring a recovery skill from environmental interaction (opening a drawer before picking). Then, ReSYNC discovers abstract concepts that ground recovery skills in a new abstract planning model. At test time, ReSYNC uses abstract planning to anticipate and avoid failures.

(pull the drawer) to resolve a failed `Pick`, and revises its planning abstractions by discovering new concepts such as `IsOpen` that modify the preconditions of existing skills. After learning, the robot not only recovers from the same failure if it occurs again, but also generalizes to a combinatorial number of unseen failures by recomposing the new skills and concepts that it has discovered.

From the perspective of abstract planning, prior work typically assumes hand-designed skills for learning concepts [7–9], hand-designed concepts for learning skills [10, 11], or substantial supervision when learning both [12]. ReSYNC relaxes these assumptions by using failure-recovery *experience*: failures expose missing skills, while recoveries generate self-supervised data for discovering concepts that make these skills reusable. From the perspective of reinforcement learning (RL) [5, 13–15], ReSYNC is a model-based, hierarchical approach that leverages symbolic structure and significant inductive bias to solve a broad distribution of long-horizon sparse-reward tasks efficiently. In both cases, our key idea is to *use recovery to drive progressive skill and concept discovery*.

We evaluate ReSYNC across four physical environments with continuous state and action spaces [16]. Our results show that ReSYNC achieves an average success rate of 70% on compositionally novel tasks, outperforming state-of-the-art recovery (and predicate) learning [3, 9], policy learning [17, 18], VLM planning [19], and hierarchical reinforcement learning (HRL) baselines [5]. We further demonstrate sim-to-real transfer of ReSYNC, where it learns a non-prehensile manipulation skill for recovery and generalizes to new scenarios via discovered concepts. Overall, these results suggest that recovery-driven skill and concept discovery is a promising approach for scaling robots to big worlds.

2 Related Work

Learning Abstractions for Hierarchical Planning: Hierarchical planners [20, 21] address long-horizon tasks using abstractions such as predicates, skills, and operators to decompose problems into tractable sub-goals. Recent work [10, 12, 22–25] studies learning these abstractions from data to reduce manual engineering. ReSYNC is particularly motivated by recent effect-centric predicate discovery frameworks [9, 25], which learn relational predicates grounded by neural networks while assuming a fixed skill library and an offline demonstration dataset. In contrast, ReSYNC jointly synthesizes both skills and concepts directly from task-driven interactions without explicit supervision.

Hierarchical Reinforcement Learning: HRL commonly adopts the options framework [26] to discover temporal abstractions and optimize high-level policies. Representative approaches include intrinsic skill discovery [5, 14], option-critic methods [27, 28], and contrastive skill learning [29, 30]. Deep Skill Graphs (DSG) [4, 5] is particularly related, as it expands a planning graph by discovering new low-level skills. Unlike DSG, ReSYNC learns relational predicates that optimize an explicit abstract world model, leading to improved planning and sample efficiency.

Integrated RL and Planning: A growing body of work combines reinforcement learning with symbolic planning. Prior methods construct reachability graphs from replay buffers [31, 32], learn RL policies to guide planning [33], or discover new skills on planning graphs [11]. More closely related are recovery learning methods [3, 6, 34–36], which use RL to recover from states where planning fails under novel conditions. However, existing approaches are largely reactive: recovery skills are triggered only after failure and do not modify the underlying abstractions. In contrast, ReSYNC discovers new concepts that ground the recovery experience, enabling skill reuse in unseen tasks.

3 Background and Problem Setting

We mainly follow the notation system from previous work [9]; see Appendix A for a glossary.

Environment Modeling: We model the environment as a Relational Factored Markov Decision Process (RF-MDP). The state $\mathbf{x} \in \mathcal{X}$ is factored by typed objects $\mathtt{o} \in \mathcal{O}$ with continuous features $\mathbf{x}_{\mathtt{o}}$ (e.g., 6D pose). A sparse-reward task $T = \langle \mathcal{O}, \mathbf{x}_0, g \rangle$ specifies objects, an initial state, and a goal $g \subseteq \mathcal{X}$. A solution is a sequence of low-level actions $\mathbf{a} \in \mathcal{A}$ that leads to the goal. During training, the robot interacts with a resettable dynamics model $f(\mathbf{x}' \mid \mathbf{x}, \mathbf{a})$ and has access to a broad initial state distribution $P(\mathbf{x}_0)$; at test time, it must achieve g in one finite-horizon rollout.

Planning Abstractions and Execution: The robot uses concepts (predicates, Ψ) and skills ($\mathcal{C}$) to solve long-horizon tasks with sparse rewards. A predicate $\psi \in \Psi$ has a type signature and classifier θ_ψ; a ground predicate $\underline{\psi}$ binds ψ to objects with truth value $\theta_{\underline{\psi}}(\mathbf{x})$. Each skill $\mathtt{C} \in \mathcal{C}$ has a type signature, a policy $\pi_{\mathtt{C}}$, and a terminal function $\beta_{\mathtt{C}}$; a ground skill $\underline{\mathtt{C}}$ binds it to objects. Following prior works [9, 25], we associate skills $\mathcal{C}$ with an operator set $\mathtt{Op}^{\mathcal{C}}$ using Planning Domain Definition Language (PDDL) [37]; additional details see Appendix E. Given $\mathbf{x}_0$ and g, abstract planning maps them into symbolic representations via Ψ, searches for ground skills using $\mathtt{Op}^{\mathcal{C}}$ [38], executes each skill until termination, and replans when observed effects deviate. We denote this integrated planning and execution process as $\mathrm{Plan}_f(\mathbf{x} \mid \mathcal{C}, \Psi, \mathtt{Op}^{\mathcal{C}}, g)$, yielding the final state after executing the last skill.

Initial Knowledge: The robot starts with initial skills $\mathcal{C}_0$, concepts Ψ_0, and operators $\mathtt{Op}^{\mathcal{C}_0}$ that solve simple tasks, such as relocating an unobstructed hammer (Figure 2). However, these initial abstractions are often oblivious to complexity; for example, they may implicitly assume an object is always reachable, failing to account for closed drawers or obstructing blocks.

Training Curriculum: A user-specified curriculum presents related tasks in stages to elicit failures. Let $\mathcal{T}_i^{\mathtt{train}}$ be the i^{th} stage with $N_i = 10$ tasks in our experiments. We assume the robot could detect failure states (e.g., simulator collisions) [3] and identify responsible objects through environmental interaction. For simplicity, we next describe the method assuming a single failure object per stage, denoted $\mathtt{o}_i^{\mathcal{F}}$; in the running example (Figure 2), $\mathtt{o}_1^{\mathcal{F}} = \mathtt{drawer}$ and $\mathtt{o}_2^{\mathcal{F}} = \mathtt{block}$. See Section 5 and Appendix E for situations with multiple failure objects.

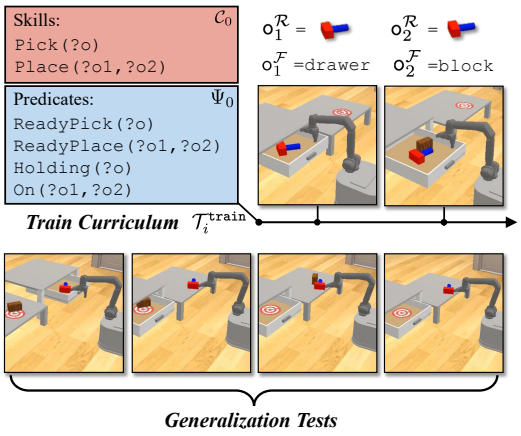

Figure 2: Overview of the running example (Cluttered Drawer). We show the initial knowledge available to the robot, each stage of the curriculum, and example generalization tests.

Test Time: After training, the robot is evaluated on held-out tasks with different and more objects than those seen during training. We measure success as achieving the task goal within a finite horizon. A naive robot that ignores the training curriculum frequently fails at test time, while merely learning recovery skills and failure detectors is still insufficient for generalization [3]. For example, after learning to open a drawer to recover from a failed grasp, the robot should also infer that it must first put down a held

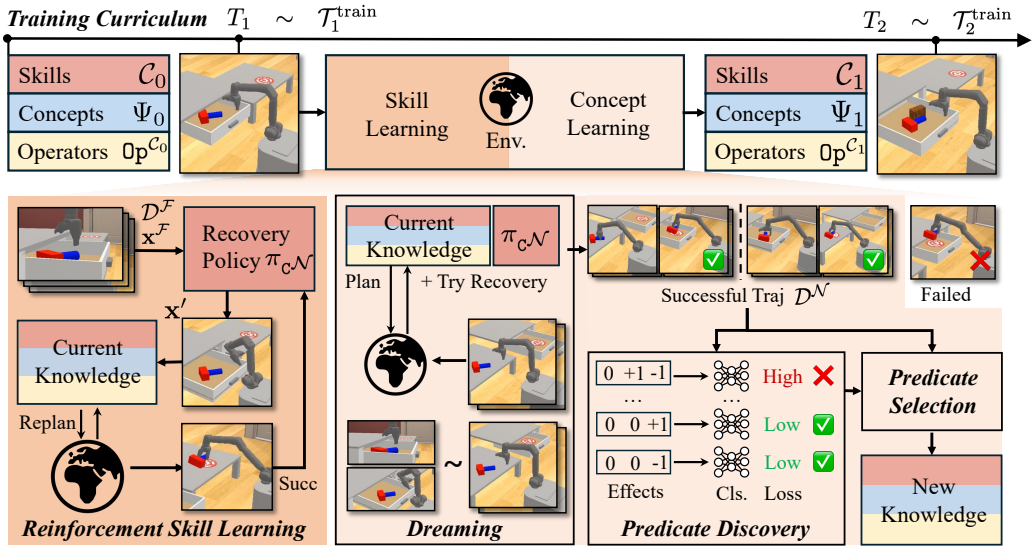

Figure 3: The ReSYNC framework. **Top:** ReSYNC alternates between recovery skill learning and concept discovery in each stage. **Bottom left:** Skill learning uses successful replanning as the recovery reward. **Bottom right:** Dreaming generates trajectories for predicate and operator learning.

object before opening the drawer in a new context. Thus, beyond skills, the robot must also learn concepts that enable reasoning over skill composition for test-time generalization.

4 Methodology

We now present ReSYNC, our method for recovery-driven skill and concept synthesis (Figure 3). **Overview:** ReSYNC learns from a staged curriculum by progressively augmenting its skill and concept library. At each stage, the robot plans until failure, mines related failures, and learns an RL recovery skill that reaches states from which planning can resume (Section 4.1). It then uses skill-generated data to discover predicates and update operators, making the new skill composable by the planner (Section 4.2). The updated library is reused in later stages (Section 4.3) and, at test time, supports planning on unseen tasks. Unless otherwise specified, we omit stage subscripts below.

4.1 Skill Learning

Failure Mining: For each training task $T_j \in \mathcal{T}^{\texttt{train}}$, ReSYNC plans and executes with its current abstractions until failure, producing $\mathbf{x}_j^{\mathcal{F}}$. Within a stage, tasks share the goal g, failure object $\texttt{o}^{\mathcal{F}}$, and failed skill $\texttt{C}^{\mathcal{F}}$; otherwise, tasks can be split into more stages. To expand the few failures $\{\mathbf{x}_j^{\mathcal{F}}\}_{j=1}^N$, ReSYNC resamples $\texttt{o}^{\mathcal{F}}$ relative to a *relevant object* $\texttt{o}^{\mathcal{R}}$ derived from $\underline{\texttt{C}}^{\mathcal{F}}$ [10]; e.g., for the hammer-picking collision with a block, $\texttt{o}^{\mathcal{F}} = \texttt{block}$ and $\texttt{o}^{\mathcal{R}} = \texttt{hammer}$. Using $\{\mathbf{x}_j^{\mathcal{F}}\}_{j=1}^N$, ReSYNC fits pairwise Gaussians over the relative states of $\texttt{o}^{\mathcal{F}}$ and $\texttt{o}^{\mathcal{R}}$, resamples $\texttt{o}^{\mathcal{F}}$ in new initial states $\mathbf{x}_0' \sim P(\mathbf{x}_0)$, and replans. It keeps only resulting failure states, yielding the augmented dataset $\mathcal{D}^{\mathcal{F}}$.

Policy and Terminal Learning: ReSYNC next constructs a *recovery learning MDP* from $\mathcal{D}^{\mathcal{F}}$ and the original task goal g. The MDP shares the same dynamics as the environment, but initializes from states in $\mathcal{D}^{\mathcal{F}}$ and defines success as reaching a state $\mathbf{x}'$ from which $\text{Plan}_f(\mathbf{x}' \mid \mathcal{C}, \Psi, \texttt{Op}^{\mathcal{C}}, g)$ succeeds. We use PPO [39] to learn a recovery policy $\pi_{\texttt{C}^{\mathcal{N}}}$ and train a terminal function $\beta_{\texttt{C}^{\mathcal{N}}}$ via supervised learning. Both the policy and the terminal function are relational [10]: given a state $\mathbf{x}$, we apply a projection $\text{proj}(\mathbf{x})$ containing only features of $\texttt{o}^{\mathcal{R}}$ and $\texttt{o}^{\mathcal{F}}$. The learned skill $\texttt{C}^{\mathcal{N}}$ can then be grounded with different objects of the same types. The library is updated with $\texttt{C}^{\mathcal{N}}$, but the planner cannot yet leverage it without a symbolic description. For that, we proceed to concept learning.

4.2 Concept Learning

A key contribution of ReSYNC is that it not only recovers from failures observed during training, but also discovers concepts that support recovery skill reuse at test time. After adding a recovery skill $\texttt{C}^{\mathcal{N}}$ to form $\mathcal{C}' = \mathcal{C} \cup \{\texttt{C}^{\mathcal{N}}\}$, ReSYNC learns predicates Ψ' and operators $\texttt{Op}^{\mathcal{C}'}$ that enable abstract planning with $\texttt{C}^{\mathcal{N}}$. For example, after learning to pull a drawer before grasping a hammer, the robot should discover `IsOpen` so the skill can later be reused before placing objects into the drawer.

Offline Concept Learning with IVNTR: To learn these abstractions, ReSYNC builds on IVNTR [9], a bilevel learning framework for jointly discovering predicate symbols and training their neural classifiers. IVNTR alternates between symbolic structure learning and neural classifier optimization, then searches over predicate subsets using a planning-efficiency surrogate objective [7]. We refer readers to Li et al. [9] and Appendix E for additional details. However, IVNTR assumes access to a large transition dataset $\mathcal{D}^{\mathcal{N}}$ that broadly covers the states encountered during planning and execution. In our setting, no demonstrations are available and new skills are learned online, motivating ReSYNC to generate its own data so that IVNTR can be applied for concept learning.

Self-Supervised Data Generation: A simple data source would be the trajectories collected during recovery skill learning (Section 4.1). By definition, this experience includes sequences of skills leading up to the new recovery skill, the recovery skill itself, and a final sequence of skills that completes the training task. However, such data only captures the original recovery context (e.g., pulling a drawer to grasp a hammer) and does not contain examples of recovery skill reuse (e.g., pulling the drawer for placing another object). In other words, we have data for $\texttt{Pull}(\texttt{hammer})$, but not for $\texttt{Pull}(\texttt{target})$. The latter data is important for disambiguating `IsOpen` from other possible explanations of successful recovery, e.g., `HammerInOpenDrawer`.

To collect reuse examples, ReSYNC performs *dreaming*, a self-supervised data generation process. As in failure mining, dreaming creates counterfactual tasks by resampling $\texttt{o}^{\mathcal{F}}$ relative to other objects. The robot plans in these tasks, invokes the new recovery skill when failure occurs, resumes planning after termination, and adds successful transitions to the offline concept-learning dataset.

Dreaming differs in how it resamples initial states: it targets states *before and after skill reuse*, such as states before and after executing $\texttt{Pull}(\texttt{target})$. Recall that we previously fit pairwise Gaussian distributions to capture the relative states between $\texttt{o}^{\mathcal{F}}$ and $\texttt{o}^{\mathcal{R}}$. These distributions correspond to states *before* the *original* recovery skill (e.g., $\texttt{Pull}(\texttt{hammer})$). To generate data that capture skill reuse on other objects, we additionally fit pairwise Gaussian distributions after the original recovery skill. We then resample $\texttt{o}^{\mathcal{F}}$'s state from one of the two pairwise Gaussian distributions, now using additional reference objects beyond the original $\texttt{o}^{\mathcal{R}}$. For instance, sampling an open `drawer` near a `target` creates data where the drawer is open but empty, separating `IsOpen` from `HammerInOpenDrawer`.

4.3 Progressive Learning

After the first stage, ReSYNC enters later stages with the latest skills, concepts, and operators, progressively expanding its library for increasingly complex unseen tasks. Two designs support multi-stage concept learning and refinement: compositional dreaming and concept fine-tuning.

Compositional Dreaming: As training progresses, the dreaming process becomes compositional. Failure objects $\texttt{o}_v^{\mathcal{F}}, v \in \{1, \ldots, i\}$ from previous stages, together with their learned state distributions, are combinatorially composed with objects matching the types of $\texttt{o}_v^{\mathcal{R}}$. This process produces increasingly diverse trajectories that improve coverage for concept discovery.

Concept Fine-tuning: A simple strategy to manage the concepts would be relearning them from scratch after each stage using all of the data, but this is computationally inefficient. Instead, ReSYNC retains previously learned concepts and incrementally expands the abstraction library using newly generated data. However, freezing existing concepts can lead to inconsistencies when newly learned skills alter previously observed state distributions. For example, a recovery skill that pushes a `block` away from a `hammer` may perturb a nearby `drawer`, causing the predicate `IsOpen` to become inaccurate. To address this issue, ReSYNC fine-tunes the classifiers for existing concepts before

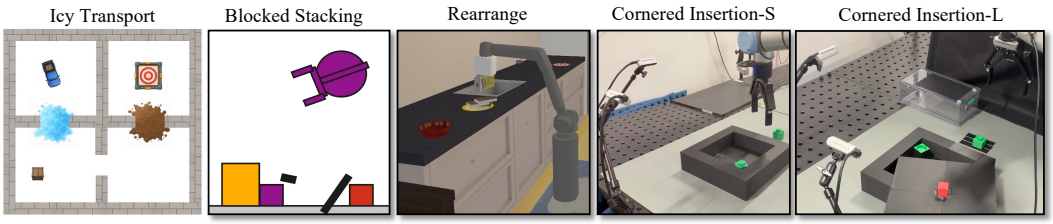

Figure 4: Visualization of the three simulated domains (excluding Cluttered Drawer) and the two real-world domains. For visualization of the skills, please refer to our attached supplementary video.

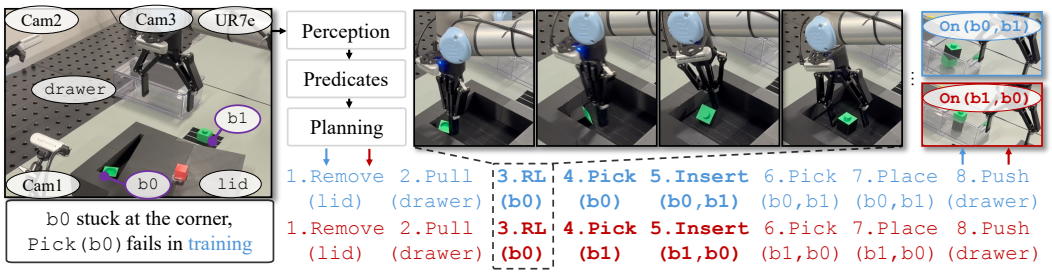

Figure 5: Hardware setup and ReSYNC pipeline in the real-world experiments. During training (blue), the robot fails to pick b0 and learns an RL skill for recovery. In unseen test tasks with a different goal (red), ReSYNC plans to first use the RL skill on b0 to avoid the insertion failure.

discovering new ones, ensuring that they remain consistent with the expanded skill set and the newly observed states. In particular, we extend IVNTR to handle the case where some symbolic concepts are known but others must be discovered (and all classifiers must be trained). See Appendix E.

5 Experiments

We use experiments to answer the following questions about ReSYNC's efficacy and efficiency:

- **Q1.** Can ReSYNC recover from failures and generalize?
- **Q2.** Does ReSYNC progressively learn without forgetting?
- **Q3.** Can ReSYNC be deployed in the real world for planning and recovery?
- **Q4.** Do compositional dreaming and fine-tuning help?
- **Q5.** To what extent does ReSYNC perform sample-efficient learning during training time?

5.1 Implementation Details

Hardware: Simulated experiments were conducted on an NVIDIA A100 GPU with an AMD EPYC 7543 CPU. For real experiments, models were trained on an NVIDIA RTX PRO 5000 GPU and evaluated on an NVIDIA RTX 3090 GPU. In Figure 5, the real setup uses a UR7e arm with a Robotiq 2F-140 gripper and three RealSense D435i cameras providing side, front, and wrist views.

Baselines: To ensure fairness, all baselines are given the same initial knowledge and learned recovery skills. We compare ReSYNC against Recovery Chaining (RC) [3], RC+IVNTR [9], Deep Skill Graph single-task (DSG-S) and multi-task (DSG-M) [4, 5], a VLM planner with one in-context example (VLM-OS, GPT-5.4 [19]), and supervised GNN-Policy and TF-Policy [17, 18] trained on $\mathcal{D}^{\mathcal{N}}$. Additional details and analysis are provided in Appendix C.

Domains: We study four simulated domains and two real-world domains (Figure 2, Figure 4) involving long-horizon physical interactions; details are deferred to Appendix D.

- *Icy Transport (Sim):* A 2D cargo transport task [40] where icy or muddy doorway regions induce stochastic disturbances, requiring stable transport and generalization to unseen layouts.

Domain	Icy Transport					Cluttered Drawer				
Test Dist.	$\mathcal{T}_1^{\text{train}}$	$\mathcal{T}_1^{\text{test}}$	$\mathcal{T}_2^{\text{train}}$	$\mathcal{T}_2^{\text{test}}$	$\mathcal{T}_1$	$\mathcal{T}_1^{\text{train}}$	$\mathcal{T}_1^{\text{test}}$	$\mathcal{T}_2^{\text{train}}$	$\mathcal{T}_2^{\text{test}}$	$\mathcal{T}_1$
Horizon	3000	3000	5000	5000	5000	500	500	800	800	800
GNN [17]	**0.98**$_{\pm.03}$	0.01$_{\pm.01}$	0.18$_{\pm.13}$	0.18$_{\pm.13}$	0.38$_{\pm.18}$	0.86$_{\pm.04}$	0.00$_{\pm.00}$	**0.69**$_{\pm.07}$	0.48$_{\pm.10}$	0.00$_{\pm.00}$
TF [18]	**0.98**$_{\pm.03}$	0.00$_{\pm.00}$	0.26$_{\pm.14}$	0.17$_{\pm.12}$	0.52$_{\pm.16}$	0.11$_{\pm.07}$	0.00$_{\pm.00}$	0.03$_{\pm.03}$	0.02$_{\pm.02}$	0.00$_{\pm.00}$
VLM-OS [19]	0.39$_{\pm.13}$	0.00$_{\pm.00}$	0.30$_{\pm.14}$	0.00$_{\pm.00}$	0.24$_{\pm.12}$	0.68$_{\pm.06}$	0.29$_{\pm.03}$	0.18$_{\pm.04}$	0.25$_{\pm.06}$	0.50$_{\pm.10}$
DSG-S [5]	0.67$_{\pm.12}$	0.00$_{\pm.00}$	0.24$_{\pm.07}$	0.02$_{\pm.02}$	0.44$_{\pm.08}$	0.73$_{\pm.03}$	0.04$_{\pm.01}$	0.67$_{\pm.04}$	0.04$_{\pm.01}$	0.38$_{\pm.02}$
DSG-M [5]	0.61$_{\pm.04}$	0.33$_{\pm.04}$	0.15$_{\pm.07}$	0.01$_{\pm.02}$	0.21$_{\pm.05}$	0.13$_{\pm.02}$	0.03$_{\pm.01}$	0.00$_{\pm.00}$	0.00$_{\pm.00}$	0.09$_{\pm.04}$
RC [3]	0.71$_{\pm.11}$	0.00$_{\pm.00}$	0.34$_{\pm.16}$	0.00$_{\pm.00}$	0.50$_{\pm.18}$	0.68$_{\pm.03}$	0.01$_{\pm.01}$	0.62$_{\pm.04}$	0.14$_{\pm.02}$	0.35$_{\pm.15}$
RC+IVNTR [9]	0.50$_{\pm.20}$	0.44$_{\pm.18}$	0.18$_{\pm.12}$	0.08$_{\pm.08}$	0.66$_{\pm.16}$	0.69$_{\pm.03}$	0.59$_{\pm.15}$	0.64$_{\pm.03}$	0.27$_{\pm.18}$	0.58$_{\pm.14}$
ReSYNC	0.92$_{\pm.05}$	**0.76**$_{\pm.16}$	**0.85**$_{\pm.09}$	**0.68**$_{\pm.13}$	**0.87**$_{\pm.09}$	**0.95**$_{\pm.01}$	**0.96**$_{\pm.02}$	0.64$_{\pm.02}$	**0.72**$_{\pm.03}$	**0.76**$_{\pm.03}$

Domain	Blocked Stacking								Rearrange	
Test Dist.	$\mathcal{T}_1^{\text{train}}$	$\mathcal{T}_1^{\text{test}}$	$\mathcal{T}_2^{\text{train}}$	$\mathcal{T}_2^{\text{test}}$	$\mathcal{T}_1$	$\mathcal{T}_3^{\text{train}}$	$\mathcal{T}_3^{\text{test}}$	$\mathcal{T}_{1,2}$	$\mathcal{T}_1^{\text{train}}$	$\mathcal{T}_1^{\text{test}}$
Horizon	200	200	400	400	400	500	500	500	5000	5000
GNN [17]	0.74$_{\pm.03}$	0.08$_{\pm.06}$	0.55$_{\pm.11}$	0.12$_{\pm.10}$	0.00$_{\pm.00}$	0.56$_{\pm.15}$	0.07$_{\pm.06}$	0.03$_{\pm.02}$	0.39$_{\pm.12}$	0.00$_{\pm.00}$
TF [18]	0.74$_{\pm.03}$	0.05$_{\pm.10}$	0.66$_{\pm.05}$	0.11$_{\pm.12}$	0.68$_{\pm.02}$	**0.72**$_{\pm.05}$	0.08$_{\pm.06}$	0.35$_{\pm.16}$	0.15$_{\pm.14}$	0.00$_{\pm.00}$
VLM-OS [19]	0.74$_{\pm.01}$	0.04$_{\pm.03}$	0.19$_{\pm.15}$	0.02$_{\pm.01}$	0.21$_{\pm.10}$	0.00$_{\pm.00}$	0.09$_{\pm.08}$	0.23$_{\pm.14}$	0.26$_{\pm.02}$	0.00$_{\pm.00}$
DSG-S [5]	0.66$_{\pm.05}$	0.04$_{\pm.03}$	0.52$_{\pm.14}$	0.01$_{\pm.01}$	0.35$_{\pm.19}$	0.68$_{\pm.04}$	0.00$_{\pm.00}$	0.17$_{\pm.07}$	0.35$_{\pm.03}$	0.00$_{\pm.00}$
DSG-M [5]	0.38$_{\pm.06}$	0.05$_{\pm.03}$	0.01$_{\pm.02}$	0.01$_{\pm.01}$	0.07$_{\pm.10}$	0.00$_{\pm.00}$	0.05$_{\pm.03}$	0.05$_{\pm.02}$	0.18$_{\pm.03}$	0.15$_{\pm.00}$
RC [3]	**0.75**$_{\pm.03}$	0.04$_{\pm.04}$	0.61$_{\pm.11}$	0.24$_{\pm.18}$	0.39$_{\pm.17}$	0.10$_{\pm.03}$	0.16$_{\pm.12}$	0.17$_{\pm.14}$	0.00$_{\pm.00}$	0.00$_{\pm.00}$
RC+IVNTR [9]	0.74$_{\pm.02}$	0.70$_{\pm.03}$	0.64$_{\pm.04}$	0.17$_{\pm.13}$	0.72$_{\pm.02}$	0.64$_{\pm.04}$	0.37$_{\pm.17}$	0.09$_{\pm.10}$	0.47$_{\pm.12}$	0.48$_{\pm.12}$
ReSYNC	0.73$_{\pm.04}$	**0.76**$_{\pm.05}$	**0.67**$_{\pm.10}$	**0.84**$_{\pm.06}$	**0.76**$_{\pm.02}$	0.69$_{\pm.05}$	**0.72**$_{\pm.08}$	**0.80**$_{\pm.06}$	**0.59**$_{\pm.03}$	**0.57**$_{\pm.03}$

Table 1: Success rate comparison on the four simulated domains. Our ReSYNC achieves the best results in unseen tests and competitive results in seen tasks. ReSYNC also progressively learns from multiple stages without forgetting. Results are averaged over five random seeds ($\pm$ stdev).

Tasks	Cornered Insertion-S ($\mathcal{T}_1^{\text{train}}$|$\mathcal{T}_1^{\text{test}}$)								Cornered Insertion-L ($\mathcal{T}_1^{\text{train}}$|$\mathcal{T}_1^{\text{test}}$)							
Seed	S0	S1	S2	AVG	S0	S1	S2	AVG	S0	S1	S2	AVG	S0	S1	S2	AVG
Human	0.9	0.7	0.8	0.80	0.7	0.9	0.8	0.80	0.8	0.8	0.6	0.73	0.8	0.7	0.8	0.77
RC	0.8	0.8	0.7	**0.77**	0.0	0.0	0.0	0.00	0.7	0.7	0.5	0.63	0.0	0.0	0.0	0.00
ReSYNC	0.7	0.8	0.7	0.73	0.6	0.8	0.9	**0.77**	0.6	0.8	0.6	**0.67**	0.5	0.6	0.8	**0.63**

Table 2: Quantitative results for real world domains. In $\mathcal{T}_1^{\text{train}}$, both ReSYNC and RC [3] perform comparably. In $\mathcal{T}_1^{\text{test}}$, ReSYNC uses learned predicates to avoid failure, yielding better results.

- *Blocked Stacking (Sim):* A 2D robot [40] stacks blocks while clearing static and movable obstructions.
- *Cluttered Drawer (Sim):* A 3D mobile manipulator retrieves a hammer from cluttered scenes with drawers and obstructions, implemented in ManiSkill [16].
- *Rearrange (Sim):* Adapted from ManiSkill-HAB [16], this domain requires rearranging objects despite several large obstructions at target locations, demanding tool-use skills.
- *Cornered Insertion-S (Real):* A UR7e inserts Lego blocks into a tower. One block begins trapped in a box corner, requiring learned non-prehensile manipulation before insertion.
- *Cornered Insertion-L (Real):* A longer version of "Cornered Insertion-S", where the robot additionally needs to remove a lid from the box and place the tower into a drawer.

Experiment Setup: Each domain follows a staged curriculum for skill and concept acquisition. After each stage, we evaluate on: (i) $\mathcal{T}_i^{\text{train}}$ tasks sampled from the training distribution; (ii) $\mathcal{T}_i^{\text{test}}$ tasks generated by swapping object-centric states among same-type objects; and (iii) $\mathcal{T}_{i-1}$ tasks from earlier stages. We report mean success rates and standard deviations over 5 seeds, averaged across 50 tasks per scenario and all scenarios within each category; detailed results and explanations are in Appendix B. For real-world domains, we follow OmniReset [41] to train recovery policies in IsaacSim with domain randomization [42, 43], then distill them into RGB policies [44] using calibrated three-view observations. Predicates are learned from simulated object poses. The robot initially scans the scene with its wrist camera and SAM3 [45] to obtain object-centric states and generates a high-level plan with learned predicates, and then sequentially executes learned and given skills. For both $\mathcal{T}_i^{\text{train}}$ and $\mathcal{T}_i^{\text{test}}$ scenarios, we evaluate 10 randomly initialized tasks over 3 seeds.

5.2 Empirical Results

Simulated Domains: We report average success rates across all four domains in Table 1. On $\mathcal{T}_i^{\text{train}}$ tasks, ReSYNC successfully recovers and performs competitively with or better than the baselines (**Q1**). ReSYNC shows three strengths: (1) by leveraging learned relational abstractions, ReSYNC significantly outperforms baselines by $20-70\%$ on $\mathcal{T}_i^{\text{test}}$ scenarios (**Q1**). (2) ReSYNC is able to learn from multiple training stages without a significant performance drop as task horizon and variance grow (**Q2**). (3) ReSYNC does not forget the previous tasks when progressively learning from new experiences (**Q2**). We present detailed analysis of baseline failures in Appendix C.

Real Domains: As shown in Figure 5, ReSYNC learns a non-prehensile recovery skill from failed picking attempts and discovers new predicates that ground this skill. These predicates allow ReSYNC to anticipate that "Insert" will fail unless the block is first repositioned, enabling proactive recovery before insertion. In contrast, reactive recovery [3] waits until failure occurs, at which point recovery can no longer restore a solvable state. Thus, although both methods recover from the training failure, only ReSYNC generalizes the learned skill as a proactive planning operator (Table 2). For reference, we also report a "Human" baseline in which a human specifies the next skill after each previous skill. Representative failure cases are shown in Appendix I. To our knowledge, this is the first real-world demonstration of abstract planning with a learned non-prehensile RL skill and discovered predicates (**Q3**).

Metric	$J_0^{\times 1e5}$ (↓)	$J_{-1}^{\times 1e5}$	$\mathcal{T}_2^{\text{train}}$	$\mathcal{T}_2^{\text{test}}$	$\mathcal{T}_1$
✗	2.47	2.04	0.28	0.47	0.76
✓	0.48	0.03	0.67	0.84	0.76
Δ	↓ 0.81	↓ 0.99	↑ 0.46	↑ 0.42	0.00

Table 3: Ablation of concept fine-tuning in Blocked Stacking. Without fine-tuning, planning objectives inflate and test success rates drop by up to 39%.

5.3 Ablations and Analysis

Dreaming: In Table 1, RC+IVNTR is trained with the same number of trajectories as $\mathcal{D}^{\mathcal{N}}$, but drawn solely from recovery experience. In the first training stage across four domains, predicate learning without dreaming results in a 10–36% performance drop. In contrast, dreaming in ReSYNC provides a diverse trajectory distribution, yielding generalizable abstractions (**Q4**).

Finetuning Across Scenarios: As demonstrated in Table 3, omitting fine-tuning leads to "semantic drift," where previously learned predicates return erroneous truth values in the new states. This results in higher planning objectives (J) and lower success rates (**Q4**).

Sample Efficiency: We compare ReSYNC with Test-Time Recovery Learning (TTRL), which learns new skills for new failures at test time. As shown in Figure 6, ReSYNC solves $\mathcal{T}_i^{\text{train}}$ tasks with $22\times$ fewer episodes by reusing learned concepts for symbolic reasoning (**Q5**). We present more results about test-time planning efficiency comparison in Appendix G.

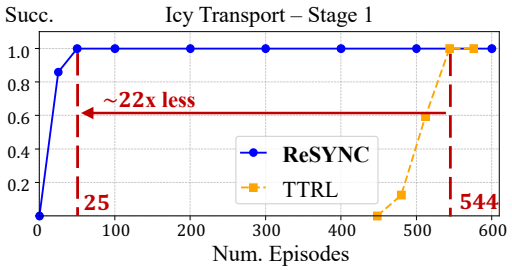

Figure 6: Learning efficiency comparison. Thanks to relational abstractions and planning, ReSYNC reduces training episodes by $\sim 22\times$.

Additional Analysis: We further discuss about the robustness of ReSYNC under different task orders and imperfect initial knowledge in Appendix F and Appendix H, respectively.

6 Discussion

Limitations: ReSYNC relies on a user-specified failure-eliciting curriculum, which could be automated with the recent agentic learning frameworks [46–48]. Dreaming uses Gaussian distributions for states, future work could study advanced generative models [49, 50] to scale ReSYNC to high-dimensional observations. ReSYNC inherits the limitations of IVNTR [9], including limited expressiveness and one-to-one skill-operator mappings. See Appendix J for more discussions.

Conclusion: We presented ReSYNC, the first method to jointly learn skills and concepts for abstract planning from environment interaction. Specifically, ReSYNC integrates recovery-skill acquisition with compositional dreaming and concept discovery, turning local reactive recovery into global proactive planning. Across four physical domains, ReSYNC substantially improves compositional generalization and sample efficiency over various prior baselines. We further demonstrate sim-to-real transfer of ReSYNC on two long-horizon manipulation tasks, where it successfully generalizes the usage of the learned non-prehensile recovery skill via discovered predicates and abstract planning.

Acknowledgments

We acknowledge the support of the Air Force Research Laboratory (AFRL), DARPA, under agreement number FA8750-23-2-1015. We also acknowledge Defence Science and Technology Agency (DSTA) under contract #DST000EC124000205. This work used Bridges-2 at PSC through allocation cis220039p from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program which is supported by NSF grants #2138259, #2138286, #2138307, #2137603, and #213296. We also acknowledge a Princeton SEAS Innovation grant. The authors would also like to express sincere gratitude to Patrick Yin (UW) for his timely support and help on our real robot domain and sim2real experiments, and to Jiayuan Mao (UPenn) for her valuable feedback on the early stages of this work.

References

[1] K. Javed and R. S. Sutton. The Big World Hypothesis and Its Ramifications for Artificial Intelligence. In *Finding the Frame: An RLC Workshop for Examining Conceptual Frameworks*, 2024.

[2] S. Vats, M. Likhachev, and O. Kroemer. Efficient Recovery Learning using Model Predictive Meta-Reasoning. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 7258–7264, 2023.

[3] S. Vats, D. K. Jha, M. Likhachev, O. Kroemer, and D. Romeres. Recoverychaining: Learning Local Recovery Policies for Robust Manipulation. *arXiv preprint arXiv:2410.13979*, 2024.

[4] A. Bagaria, J. K. Senthil, and G. Konidaris. Skill Discovery for Exploration and Planning using Deep Skill Graphs. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 521–531. PMLR, 2021.

[5] A. Bagaria, A. D. M. Koch, R. Rodriguez-Sanchez, S. Lobel, and G. Konidaris. Intrinsically Motivated Discovery of Temporally Abstract Graph-based Models of the World. In *Proceedings of Reinforcement Learning Conference*, 2025.

[6] A. Li, N. Kumar, T. Lozano-Pérez, and L. P. Kaelbling. Learning to Bridge the Gap: Efficient Novelty Recovery with Planning and Reinforcement Learning. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

[7] T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Pérez, L. Kaelbling, and J. B. Tenenbaum. Predicate Invention for Bilevel Planning. In *Proceedings of The AAAI Conference on Artificial Intelligence (AAAI)*, volume 37, pages 12120–12129, 2023.

[8] N. Shah, J. Nagpal, P. Verma, and S. Srivastava. From Reals to Logic and Back: Inventing Symbolic Vocabularies, Actions and Models for Planning from Raw Data. *arXiv preprint arXiv:2402.11871*, 2024. URL https://arxiv.org/pdf/2402.11871v2.

[9] B. Li, T. Silver, S. Scherer, and A. Gray. Bilevel Learning for Bilevel Planning. In *Proceedings of the Robotics: Science And Systems (RSS)*, 2025.

[10] T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. Learning Neuro-Symbolic Skills for Bilevel Planning. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2022.

[11] Y. I. Liu, B. Li, B. Eysenbach, and T. Silver. SLAP: Shortcut Learning for Abstract Planning. *arXiv preprint arXiv:2511.01107*, 2025.

[12] Y. S. Shao, Y. Zheng, S. Sun, P. Chaudhari, V. Kumar, and N. Figueroa. Symskill: Symbol and Skill Co-Invention for Data-Efficient and Real-Time Long-Horizon Manipulation. *arXiv preprint arXiv:2510.01661*, 2025.

[13] D. Abel, D. Arumugam, L. Lehnert, and M. Littman. State abstractions for lifelong reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2018.

[14] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

[15] R. K. Nayyar and S. Srivastava. Autonomous Option Invention for Continual Hierarchical Reinforcement Learning and Planning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 39, pages 19642–19650, 2025.

[16] A. Shukla, S. Tao, and H. Su. Maniskill-hab: A benchmark for low-level manipulation in home rearrangement tasks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

[17] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational Inductive Biases, Deep Learning, and Graph Networks. *arXiv preprint arXiv:1806.01261*, 2018.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is All You Need. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

[19] OpenAI. GPT-5.4 Thinking System Card, Mar. 2026. URL https://openai.com/index/gpt-5-4-thinking-system-card/.

[20] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical Task and Motion Planning in the Now. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 1470–1477. IEEE, 2011.

[21] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. Pddlstream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 30, pages 440–448, 2020.

[22] R. Chitnis, T. Silver, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. Learning Neuro-Symbolic Relational Transition Models for Bilevel Planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4166–4173, 2022.

[23] N. Kumar, T. Silver, W. McClinton, L. Zhao, S. Proulx, T. Lozano-Pérez, L. P. Kaelbling, and J. Barry. Practice Makes Perfect: Planning To Learn Skill Parameter Policies. In *Proceedings of the Robotics: Science And Systems (RSS)*, 2024.

[24] G. Konidaris, L. P. Kaelbling, and T. Lozano-Pérez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research (JAIR)*, 2018.

[25] Q. Wang, B. Li, Z. Luo, Y. Xu, A. Gray, T. Silver, S. Scherer, K. Sycara, and Y. Xie. Unifying Deep Predicate Invention with Pre-trained Foundation Models. *arXiv preprint arXiv:2512.17992*, 2025.

[26] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial intelligence*, 1999.

[27] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 31, 2017.

[28] S. Zhang and S. Whiteson. DAC: The Double Actor-Critic Architecture for Learning Options. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

[29] S. Moon, J. Yeom, B. Park, and H. O. Song. Discovering Hierarchical Achievements in Reinforcement Learning via Contrastive Learning. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 63674–63686, 2023.

[30] B. Eysenbach, T. Zhang, S. Levine, and R. R. Salakhutdinov. Contrastive Learning as Goal-Conditioned Reinforcement Learning. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 35603–35620, 2022.

[31] B. Eysenbach, R. Salakhutdinov, and S. Levine. Search on the Replay Buffer: Bridging Planning and Reinforcement Learning. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

[32] N. Savinov, A. Dosovitskiy, and V. Koltun. Semi-Parametric Topological Memory for Navigation. *arXiv preprint arXiv:1803.00653*, 2018.

[33] F. Yang, D. Lyu, B. Liu, and S. Gustafson. PEORL: Integrating Symbolic Planning and Hierarchical Reinforcement Learning for Robust Decision-Making. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4860–4866, 2018.

[34] Y. Jiang, F. Yang, S. Zhang, and P. Stone. Integrating Task-Motion Planning with Reinforcement Learning for Robust Decision Making in Mobile Robots. *arXiv preprint arXiv:1811.08955*, 2018.

[35] V. Sarathy, D. Kasenberg, S. Goel, J. Sinapov, and M. Scheutz. Spotter: Extending Symbolic Planning Operators through Targeted Reinforcement Learning. *arXiv preprint arXiv:2012.13037*, 2020.

[36] S. Goel, Y. Shukla, V. Sarathy, M. Scheutz, and J. Sinapov. Rapid-learn: A Framework for Learning to Recover for Handling Novelties in Open-world Environments. In *2022 IEEE International Conference on Development and Learning (ICDL)*, pages 15–22. IEEE, 2022.

[37] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. Sri, A. Barrett, D. Christianson, et al. Pddl—the planning domain definition language. *Technical Report, Tech. Rep.*, 1998.

[38] M. Helmert. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[39] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[40] Y. Huang, B. Li, V. Saxena, Y. Liang, U. A. Mishra, L. Ji, L. Zha, J. Wu, N. Kumar, S. Scherer, et al. KinDER: A Physical Reasoning Benchmark for Robot Learning and Planning. *arXiv preprint arXiv:2604.25788*, 2026.

[41] P. Yin, T. Westenbroek, Z. Zhang, J. Tran, I. Dagnino, E. Shilamkar, N. Mbiziwo-Tiapo, S. Bagaria, X. Liu, G. Mullins, A. Kolobov, and A. Gupta. Emergent Dexterity via Diverse Resets and Large-Scale Reinforcement Learning. In *The Fourteenth International Conference on Learning Representations*, 2026.

[42] F. Bjelonic, F. Tischhauser, and M. Hutter. Towards Bridging the Gap: Systematic Sim-to-Real Transfer for Diverse Legged Robots. *arXiv preprint arXiv:2509.06342*, 2025.

[43] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. Solving Rubik's Cube with a Robot Hand. *arXiv preprint arXiv:1910.07113*, 2019.

[44] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Robotics: Science and Systems (RSS)*, 2023.

[45] N. Carion, L. Gustafson, Y.-T. Hu, S. Debnath, R. Hu, D. Suris, C. Ryali, K. V. Alwala, H. Khedr, A. Huang, et al. SAM 3: Segment Anything with Concepts. *arXiv preprint arXiv:2511.16719*, 2025.

[46] P. Liu, L. P. Kaelbling, J. B. Tenenbaum, and J. Mao. Lifelong Experience Abstraction and Planning. In *ICML 2025 Workshop on Programmatic Representations for Agent Learning*, 2025.

[47] M. Fu, J. Yu, K. El-Refai, E. Kou, H. Xue, H. Huang, W. Xiao, G. Wang, F.-F. Li, G. Shi, et al. CaP-X: A Framework for Benchmarking and Improving Coding Agents for Robot Manipulation. *arXiv preprint arXiv:2603.22435*, 2026.

[48] R. Zabounidis, Y. Wu, S. Stepputtis, W. Kim, Y. Li, T. Mitchell, and K. Sycara. SCALAR: Learning and Composing Skills through LLM Guided Symbolic Planning and Deep RL Grounding. *arXiv preprint arXiv:2603.09036*, 2026.

[49] J. Luo, T. Ding, K. H. R. Chan, H. Min, C. Callison-Burch, and R. Vidal. Concept Lancet: Image Editing with Compositional Representation Transplant. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 28502–28512, 2025.

[50] J. Luo, J. Yang, T. Neiman, L. Fan, B. Yin, S. Tran, M. Shah, and R. Vidal. Dictionary-Aligned Concept Control for Safeguarding Multimodal LLMs. *arXiv preprint arXiv:2604.08846*, 2026.

[51] A. Athalye, N. Kumar, T. Silver, Y. Liang, T. Lozano-Pérez, and L. P. Kaelbling. Predicate Invention from Pixels via Pretrained Vision-Language Models. *arXiv preprint arXiv:2501.00296*, 2024.

[52] N. Kumar, W. Shen, F. Ramos, D. Fox, T. Lozano-Pérez, L. P. Kaelbling, and C. R. Garrett. Open-World Task and Motion Planning via Vision-Language Model Generated Constraints. *IEEE Robotics and Automation Letters (RA-L)*, 11:3366–3373, 2026. URL https://arxiv.org/abs/2411.08253.

[53] J. Duan, W. Pumacay, N. Kumar, Y. R. Wang, S. Tian, W. Yuan, R. Krishna, D. Fox, A. Mandlekar, and Y. Guo. AHA: A Vision-Language-Model for Detecting and Reasoning Over Failures in Robotic Manipulation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2410.00371.

[54] S. Wang, M. Han, Z. Jiao, Z. Zhang, Y. N. Wu, S.-C. Zhu, and H. Liu. Llm^3:large language model-based task and motion planning with motion failure reasoning. *arXiv preprint arXiv:2403.11552*, 2024. URL https://arxiv.org/pdf/2403.11552.

[55] C. Xu, T. K. Nguyen, E. Dixon, C. Rodriguez, P. Miller, R. Lee, P. Shah, R. Ambrus, H. Nishimura, and M. Itkina. Can we detect failures without failure data? uncertainty-aware runtime failure detection for imitation learning policies. In *Proceedings of the Robotics: Science And Systems (RSS)*, 2025.

[56] Z. Yang, B. Hedegaard, A. Jaafar, Y. Wei, S. Thompson, S. S. Raman, H. Fu, S. Tellex, G. Konidaris, D. Paulius, et al. SkillWrapper: Generative Predicate Invention for Skill Abstraction. *arXiv preprint arXiv:2511.18203*, 2025.

[57] Z. Yang, J. Mao, Y. Du, J. Wu, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. Compositional Diffusion-Based Continuous Constraint Solvers. In *Conference on Robot Learning (CoRL)*, 2023. URL https://arxiv.org/pdf/2309.00966.pdf.

Table 4: Important notations used in this work.

Symbol	Description
$\mathbf{x} \in \mathcal{X}$	Continuous world state; state space
$\mathbf{x}_{\mathtt{o}}$	Feature vector of object $\mathtt{o}$
$\mathbf{x}_0$	Initial state of a task
$\mathcal{O}$	Set of objects in the environment
$\mathtt{o}$	Individual object
$\mathbf{a} \in \mathcal{A}$	Low-level action; action space
$f(\mathbf{x}' \mid \mathbf{x}, \mathbf{a})$	Environment dynamics
$P(\mathbf{x}_0)$	Initial state distribution
$g \subseteq \mathcal{X}$	Goal specification
$T = \langle \mathcal{O}, \mathbf{x}_0, g \rangle$	Task definition
$\mathcal{T}_i^{\texttt{train}}$	Training tasks at curriculum stage i
N_i	Number of training tasks in stage i
$\psi \in \Psi$	Lifted predicate (concept)
$\underline{\psi}$	Ground predicate
θ_ψ	Predicate classifier
$\mathtt{C} \in \mathcal{C}$	Lifted skill
$\underline{\mathtt{C}}$	Ground skill
$\pi_{\mathtt{C}}$	Skill policy
$\beta_{\mathtt{C}}$	Skill termination function
$\mathtt{Op}^{\mathcal{C}}$	Operator set induced from skills

Symbol	Description
$\mathrm{Plan}_f(\cdot)$	Integrated planning and execution procedure
i	Curriculum stage index
T_j	j-th training task in a stage
$\mathbf{x}_j^{\mathcal{F}}$	State immediately preceding failure in T_j
$\underline{\mathtt{C}}^{\mathcal{F}}$	Ground skill active at failure
$\mathcal{C}_i$	Skill set after stage i
Ψ_i	Predicate set after stage i
$\mathtt{o}^{\mathcal{F}}$	Failure-causing object
$\mathtt{o}^{\mathcal{R}}$	Recovery-relevant object
$\mathcal{D}^{\mathcal{F}}$	Mined failure-state dataset
$\mathbf{x}_0'$	Resampled initial state for mining/dreaming
$\mathtt{C}^{\mathcal{N}}$	Newly learned recovery skill
$\mathcal{C}'$	Skill set after adding $\mathtt{C}^{\mathcal{N}}$
$\mathrm{proj}(\mathbf{x})$	Projection onto relevant objects
$\mathcal{D}^{\mathcal{N}}$	Dataset for concept learning
$\mathcal{D}_i^{\mathcal{N}}$	Concept-learning dataset at stage i
$\tilde{\Psi}^{\mathcal{N}}$	Candidate predicates discovered from dreamed data
$\Psi^{\mathcal{N}}$	Selected newly discovered predicates
Ψ'	Expanded predicate set after learning
$\mathtt{Op}^{\mathcal{C}'}$	Operator set after skill expansion
Δ^{ψ}	Lifted effect vector for predicate ψ
J	Planning-efficiency surrogate objective

A Notation Table

In Table 4, we present the main symbols used in Section 3, Section 4, and Section E.

B Detailed Success Rates in Scenarios

We report detailed success rates for each testing scenario in Table 5, Table 6, Table 7, and Table 8. Within each domain, scenarios are assigned unique IDs for ease of reference. As shown in the tables, ReSYNC generalizes effectively to unseen tasks (highlighted in dark gray). Moreover, due to its modular design, ReSYNC retains performance on tasks from earlier curriculum stages (highlighted in light gray), indicating minimal forgetting.

- **Blocked Stacking.** Scenario IDs describe which obstruction configurations are present while the robot stacks blocks. ID 0 contains the first-stage obstruction: a small fixed rectangle above the block to be grasped. ID 1 moves the first-stage obstruction to near target bottom block. ID 2 combines the first-stage obstruction with the second-stage movable large obstruction in a training arrangement, where both obstructions appear near the block to be grasped. IDs 3–5 are held-out combinations of the first- and second-stage obstructions, moving them to the target bottom block. IDs 6–7 isolate second-stage tasks with the movable large obstruction only. At the third stage, ID 8 is the training task involving all three obstructions, where the new thin vertical wall is placed near the bottom target block and the previous two obstructions are near the block to be grasped. ID 9 involves all the three obstructions but with the thin vertical wall moved to near the block to be grasped. IDs 10–13 are held-out combinations involving the newly introduced thin vertical wall with one of the earlier obstructions, applied to either the block to be grasped or the target bottom block. IDs 14–15 isolate the third-stage wall obstruction.
- **Icy Transport.** Scenario IDs describe which doorway perturbations the cargo robot must handle while transporting objects between rooms. Note that in this domain, the provided tasks in the same stage could involve different relevant objects. IDs 0–1 are first-stage tasks with icy doorway regions in front of one of the two transport objects. ID 2 is the held-out icy-layout scenario, where the icy region is in front of the transport target region. IDs 3–6 are the seen tasks combining first-stage icy-region with second-stage muddy-region, which

Stages	$i=1$		$i=2$							
Scenario ID	0	1	2	0	1	3	4	5	6	7
ReSYNC	0.73	**0.76**	**0.67**	**0.74**	**0.77**	**0.72**	**0.75**	**0.73**	**1.00**	**1.00**
RC	0.75	0.04	0.61	0.75	0.04	0.03	0.00	0.00	1.00	0.16
RC+IVNTR	0.74	0.70	0.64	0.74	0.70	0.00	0.00	0.61	0.00	0.23
VLM-OS	0.74	0.04	0.02	0.38	0.04	0.01	0.00	0.00	0.82	0.10
DSG-M	0.38	0.05	0.01	0.32	0.04	0.00	0.00	0.00	0.03	0.06
GNN	0.74	0.08	0.55	0.00	0.00	0.07	0.17	0.35	0.00	0.00
TF	0.74	0.05	0.66	0.72	0.77	0	0.34	0.01	0.2	0.00

Table 5: Detailed success rate in each scenario after the first and the second learning stage in the Blocked Stacking domain.

Stages	$i=3$															
Scenario ID	8	0	1	2	3	4	5	6	7	9	10	11	12	13	14	15
ReSYNC	0.69	**0.67**	0.74	**0.77**	**0.72**	**0.76**	**0.72**	**1.00**	**1.00**	**0.58**	**0.67**	**0.63**	**0.68**	**0.70**	**0.90**	0.91
RC	0.10	0.61	0.75	0.03	0.03	0.00	0.00	1.00	0.16	0.00	0.00	0.11	0.06	0.52	0.24	0.92
RC+IVNTR	0.64	0.10	0.15	0.31	0.00	0.00	0.12	0.00	0.05	0.26	0.65	0.37	0.14	0.10	0.53	0.54
VLM-OS	0.00	0.00	0.29	0.72	0.00	0.02	0.00	0.11	0.67	0.00	0.09	0.06	0.37	0.10	0.00	0.00
DSG-M	0.00	0.00	0.32	0.04	0.01	0.00	0.00	0.03	0.05	0.00	0.00	0.01	0.00	0.00	0.09	0.24
GNN	0.56	0.00	0.00	0.00	0.00	0.07	0.05	0.00	0.00	0.08	0.00	0.00	0.02	0.32	0.00	0.00
TF	0.72	0.55	0.13	0.14	0.43	0.50	0.59	0.20	0.28	0.14	0.12	0.02	0.16	0.00	0.00	0.00

Table 6: Detailed success rate in each scenario after the third learning stage in the Blocked Stacking domain.

Stages	$i=1$			$i=2$										
Scenario ID	0	1	2	3	4	5	6	0	1	2	7	8	9	10	11
ReSYNC	0.90	0.94	**0.76**	**0.74**	**0.83**	**0.89**	**0.94**	**0.91**	**0.94**	**0.76**	**0.89**	**0.56**	**0.84**	**0.40**	**0.74**
RC	0.60	0.82	0.00	0.44	0.13	0.42	0.35	0.74	0.77	0.00	0.00	0.00	0.00	0.00	0.00
RC+IVNTR	0.60	0.41	0.44	0.18	0.14	0.26	0.14	0.79	0.57	0.62	0.11	0.11	0.02	0.11	0.04
VLM-OS	0.20	0.57	0.00	0.18	0.25	0.04	0.74	0.32	0.40	0.00	0.00	0.00	0.00	0.00	0.00
DSG-M	0.58	0.64	0.33	0.00	0.00	0.37	0.22	0.30	0.23	0.09	0.07	0.00	0.00	0.00	0.00
GNN	1.00	0.96	0.00	0.00	0.30	0.26	0.15	0.30	0.34	0.50	0.00	0.09	0.00	0.27	0.56
TF	1.00	0.96	0.00	0.10	0.16	0.32	0.45	0.39	0.41	0.77	0.03	0.00	0.00	0.29	0.52

Table 7: Detailed success rate in each scenario after the first and the second learning stage in the Icy Transport domain.

are in front of one of the transport objects. IDs 7–11 are held-out mixed scenes where one of the icy/muddy regions appear in front of the transport target region, while the other appears in front of one of the transport objects.

- **Cluttered Drawer.** Scenario IDs describe drawer-and-clutter configurations for retrieving the hammer. ID 0 is the first-stage scene where the drawer is not fully open and must be pulled before the hammer can be reached. ID 1 is the held-out version, where the robot needs to place the hammer into the closed drawer. ID 2 is the second-stage training scene that combines the drawer condition with a movable block near the hammer. IDs 3–5 are held-out second-stage scenes where the block and drawer could be around the target placement region.
- **Rearrange.** Scenario IDs describe mobile-manipulation rearrangement scenes with large unpickable obstructions near target locations. ID 0 is the first-stage training distribution, where the obstructions are near the goal of the yellow-white sugar box. ID 1 is the held-out rearrangement scene, where the same obstructions are near the goal of the red bowl.

Scenario ID	Cluttered Drawer $i=1$		Cluttered Drawer $i=2$						Rearrange $i=1$	
Method	0	1	2	0	1	3	4	5	0	1
ReSYNC	**0.95**	**0.96**	0.64	**0.74**	**0.78**	**0.71**	**0.72**	**0.74**	**0.59**	**0.57**
RC	0.68	0.01	0.62	0.66	0.03	0.40	0.03	0.00	0.00	0.00
RC+IVNTR	0.69	0.59	0.64	0.57	0.59	0.00	0.00	0.81	0.47	0.48
VLM-OS	0.68	0.29	0.18	0.68	0.31	0.36	0.08	0.32	0.00	0.26
DSG-M	0.13	0.03	0.00	0.16	0.03	0.00	0.00	0.00	0.18	0.15
GNN	0.86	0.00	0.69	0.00	0.00	0.47	0.62	0.34	0.39	0.00
TF	0.11	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.15	0.00

Table 8: Detailed success rate in each scenario after each learning stage in the Cluttered Drawer and the Rearrange domain.

C Baseline Details and Analysis

Recovery Chaining (RC) [3]. RC performs passive recovery without abstract planning. At test time, plans are generated using the initial predicates and operators; when a failure detector is triggered, the corresponding learned recovery skill is executed. For fairness, RC uses the same learned failure detectors and skill policies as ReSYNC. Its main limitation is generalization: test-time failure states can differ substantially from those seen during recovery training, often making recovery ineffective. In long-horizon domains such as Icy Transport and Rearrange, RC also suffers from compounding detector errors, since the failure detector is queried at every low-level step. By contrast, ReSYNC converts the failed skill and its recovery into new planning operators, so the relevant detector is invoked only when that skill is executed, reducing compounding errors and enabling more reliable recovery execution.

RC+IVNTR [3, 9]. RC+IVNTR applies IVNTR after each RC-trained recovery skill to learn predicates and operators, but does not use dreaming or concept finetuning. Compared with RC, it can use learned recovery skills during abstract planning, improving test-time generalization and reducing detector compounding in the same way as ReSYNC. However, because concept learning only uses trajectories from the original recovery context, the learned predicates are often insufficiently robust for out-of-distribution test states. Moreover, without finetuning, predicates discovered in earlier stages may become misaligned with later skills, leading to planning failures in multi-stage domains.

Deep Skill Graph (DSG-S, DSG-M) [5]. The official implementation builds a single graph over all training tasks in each stage (DSG-S). This performs well on training-distribution tasks and often matches ReSYNC, but it cannot synthesize plan sequences absent from the training data, resulting in poor test-time generalization. We therefore also implement a multi-task variant, DSG-M, which constructs planning graphs using PDDL operators and integrates each learned recovery skill into the planner. Specifically, DSG-M introduces two predicates per skill: one for the skill's add effect and one for its precondition/delete effect. We use the same neural architectures and dreamed trajectory data as ReSYNC. However, DSG-M restricts predicates to this fixed structure, limiting expressiveness and often producing inefficient plans with redundant skill sequences. Additional quantitative results are provided in Appendix G.

VLM One-shot (VLM-OS) [19]. VLM-OS prompts GPT-5.4 [19] to generate a plan for each task in a single forward pass, without additional training or finetuning. The prompt includes the initial PDDL operators, while newly learned skill operators are given empty preconditions and effects; the VLM must infer their symbolic descriptions from in-context training examples. An example prompt for the first stage of Blocked Stacking is shown in Figure 7. VLM-OS can sometimes solve training-distribution tasks, but it struggles on test tasks because inferring precise symbolic descriptions from few examples is difficult. This issue is amplified when planning with subsemantic RL skills whose effects are hard to express in natural language, consistent with prior observations [51, 52].

VLM-OS Prompt for Blocked Stacking

````
You are an expert robot task planner. Given a scene image, a PDDL domain definition, and a PDDL problem, produce a
↪  ground operator plan that achieves the goal. The second image shows the current scene you must solve.

Carefully compare the two images to understand the objects, and their relationships, and the differences in initial
↪  states. Use the domain definition to understand the available actions, **the PDDL is incomplete** and requires you
↪  to infer the missing preconditions and effects based on the example and your understanding of the scene. Carefully
↪  inspect the image first before generating the output, **do not directly plan with the incomplete PDDL**

## PDDL Domain
```
(define (domain Domain_Scenario_1)
    (:requirements :typing)
    (:types
    block - dyn_rectangle
    obstruction_rec - kin_rectangle
    robot - kin_robot
    dynamic2d)

    (:predicates
    (Holding ?x0 - robot ?x1 - block)
    (On ?x0 - block ?x1 - block)
    (ReadyGrasp ?x0 - robot ?x1 - block)
    (ReadyPlace ?x0 - robot ?x1 - block ?x2 - block)
    )

    (:action Grasp
    :parameters (?robot - robot ?grasp_block - block)
    :precondition (and (ReadyGrasp ?robot ?grasp_block))
    :effect (and (Holding ?robot ?grasp_block)
        (not (ReadyGrasp ?robot ?grasp_block)))
    )

  (:action Place
    :parameters (?robot - robot ?grasp_block - block ?base_block - block)
    :precondition (and (Holding ?robot ?grasp_block)
        (ReadyPlace ?robot ?grasp_block ?base_block))
    :effect (and (On ?grasp_block ?base_block)
        (not (Holding ?robot ?grasp_block))
        (not (ReadyPlace ?robot ?grasp_block ?base_block)))
    )

  (:action Punch # New RL skill
    :parameters (?robot - robot ?block - block ?obstruction_rec - obstruction_rec)
    :precondition (and )
    :effect (and )
    )

  (:action ReachToGrasp
    :parameters (?robot - robot ?grasp_block - block)
    :precondition (and )
    :effect (and (ReadyGrasp ?robot ?grasp_block))
    )

  (:action ReachToGrasp_Punch # Previously failed skill
    :parameters (?robot - robot ?block - block ?obstruction_rec - obstruction_rec)
    :precondition (and )
    :effect (and )
    )

  (:action ReachToPlace
    :parameters (?robot - robot ?grasp_block - block ?base_block - block)
    :precondition (and (Holding ?robot ?grasp_block))
    :effect (and (ReadyPlace ?robot ?grasp_block ?base_block))
    )
)
```

## Solved Example
Here is a solved example from the same domain. The first image shows the example scene.

### Example Problem
```
(define (problem task)
  (:domain BlockedStacking2D-domain)
  (:objects base_block (red block) - block grasp_block (purple block) - block obstruction_rec (black small rectangle) -
 ↪  obstruction_rec robot (purple circle with arm and gripper) - robot)
  (:init )
  (:goal (and (On grasp_block base_block)))
)```

### Example Plan
```plan
(ReachToGrasp_Punch robot grasp_block obstruction_rec)
(Punch robot grasp_block obstruction_rec)
(ReachToGrasp robot grasp_block)
(Grasp robot grasp_block)
(ReachToPlace robot grasp_block base_block)
(Place robot grasp_block base_block)
```

## PDDL Problem
```
````

```
(define (problem task)
  (:domain Domain_Scenario_1)
  (:objects base_block (red block) - block grasp_block (purple block) - block obstruction_rec (black small rectangle) -
 ↪  obstruction_rec robot (purple circle with arm and gripper) - robot)
  (:init )
  (:goal (and (On grasp_block base_block)))
)```

## Output Format
Output ONLY the plan inside a ```plan``` code block, one action per line. Example:
```plan
(operator1 obj_a obj_b)
(operator2 obj_c obj_d)
```
```

Figure 7: Example VLM-OS prompt for the Blocked Stacking domain. The prompt includes the incomplete PDDL domain, one solved example, the target problem, and the required plan-only output format.

GNN/TF Policy [17, 18]. GNN/TF policies are trained offline by supervised learning on segmented dreamed trajectories, following previous work [7, 9, 25]. Given a low-level state graph $G = (V, E, u)$, where nodes represent objects, edges encode pairwise relations, and u denotes global scene features, the policy predicts both the high-level operator to execute and its object arguments. Current-state and goal atoms from the initial concept set are concatenated to node and edge features to provide symbolic context. Training minimizes a combined loss with cross-entropy for operator prediction and binary cross-entropy for object selection. We evaluate two architectures. The GNN [17] uses an Encode–Process–Decode design: MLPs embed node, edge, and global features, iterative message passing performs relational reasoning, and MLP decoders predict operator logits from the global embedding and object logits from node embeddings. The Transformer [18] flattens encoded node, edge, and global features into tokens, applies a multi-head self-attention encoder, and then unflattens the output to produce operator and object predictions with the same decoders.

D Domain Details

To start, we introduce the hyper-parameters that are shared across all domains:

- *Neural Classifiers*: All of the neural classifiers (including all of the predicate classifiers and terminal functions) learned in this work are MLPs with 4 hidden layers, with dimensions $128, 256, 256, 128$, respectively. We used GELU as activation function between layers and added 1D batch normalization. The classifiers are optimized with a learning rate of 0.001 using AdamW as optimizer.
- *Neural Policies*: All of the recovery skill policies (for all of the stages in all of the domains) take an Actor-Critic framework. Both the actor and the critic are modeled using MLPs with 4 hidden layers, with dimensions $256, 256, 256, 256$, respectively. Tanh is applied as activation function. The policy is optimized using PPO with Adam as optimizer.

This section summarizes key different implementation details for each domain. Additional details are available in the accompanying code release.

1. **Icy Transport**
   A 2D navigation domain inspired by Four Rooms [26], where a car-like robot transports objects between rooms through doorways. Doorways may contain icy or muddy regions that perturb the robot's motion. Objects can be pushed via collision.
   - *State and actions:* States are represented using SE(2) poses of all objects. Actions are 3-dimensional vectors consisting of forward throttle, steering, and torque.
   - *Object types:* Robot (`r`), transport object (`o`), target (`t`), icy region (`icy`), muddy region (`muddy`).
   - *Operators:* `GoToPickObject`(?`r`, ?`o`), `GoToPlaceObject`(?`r`, ?`o`, ?`t`).
   - *Predicates:* `HandEmpty`(?`r`), `Holding`(?`r`, ?`o`), `Delivered`(?`o`, ?`t`).

- *Hyperparameters:* In both stages 1 and 2, recovery skills are trained with a horizon of 30 and up to $500,000$ environment steps. For concept learning, we collect 600 dreamed trajectories in stage 1 and 1200 in stage 2.

Stages	Failure Objects	Failed Skill	New Skill
$i = 1$	Icy Region	`GoToPickObject`	`IcyDrive`
$i = 2$	Muddy Region	`GoToPickObject`	`MuddyDrive`

2. **Blocked Stacking.**
A 2D manipulation domain in which a robot stacks blocks while manipulating around static and dynamic obstructions. Obstruction shapes and property vary across curriculum stages.
   - *State and actions:* States are represented using SE(2) poses of all objects. Actions are 5-dimensional vectors consisting of Δx, Δy, $\Delta\theta$, arm joint delta, and finger joint delta.
   - *Object types:* Robot (`r`), blocks (`b`), obstructions (`o`).
   - *Operators:* `ReachToGrasp`(?r, ?b), `ReachToPlace`(?r, ?b), `Grasp`(?r, ?b), `Place`(?r, ?b, ?b).
   - *Predicates:* `ReadyGrasp`(?r, ?b), `ReadyPlace`(?r, ?b, ?b), `Holding`(?r, ?b), `On`(?b, ?b).
   - *Hyperparameters:* Recovery skills are trained with a horizon of 10 and up to $500,000$ environment steps at all stages. For concept learning, we collect 1000 dreamed trajectories in stage 1, 2000 in stage 2, and 6400 in stage 3.

Stages	Failure Objects	Failed Skill	New Skill
$i = 1$	A small fixed rectangle above a block	`ReachToGrasp`	`Punch`
$i = 2$	A movable large obstruction next to a block	`ReachToGrasp`	`Wiggle`
$i = 3$	A fixed thin vertical wall leaning on a block	`ReachToGrasp`	`Fiddle`

3. **Cluttered Drawer.**
A 3D mobile manipulation domain in ManiSkill, where a robot retrieves a hammer from cluttered scenes containing a closed drawer and movable blocks.
   - *State and actions:* States are represented using SE(3) poses of all objects, augmented with articulation joint positions. Actions are 10-dimensional joint-position control commands for the mobile manipulator.
   - *Object types:* Robot (`r`), hammer (`h`), drawer (`d`), block (`b`).
   - *Operators:* `BodyReachToGrasp`(?r, ?h), `BodyReachToPlace`(?r, ?h, ?h), `HandReachToGrasp`(?r, ?h), `HandReachToPlace`(?r, ?h, ?h), `Grasp`(?r, ?h), `Place`(?r, ?h, ?h).
   - *Predicates:* `BodyReadyGrasp`(?r, ?h), `HandReadyGrasp`(?r, ?h), `BodyReadyPlace`(?r, ?h, ?h), `HandReadyPlace`(?r, ?h, ?h), `Holding`(?r, ?h), `On`(?h, ?h).
   - *Hyperparameters:* Recovery skills are trained with a horizon of 16 and up to $5M$ environment steps at all stages. For concept learning, we collect 1000 dreamed trajectories in stage 1 and 1280 in stage 2.

Stages	Failure Objects	Failed Skill	New Skill
$i = 1$	The drawer is not fully open	`HandReachToGrasp`	`Pull`
$i = 2$	A block next to a hammer	`HandReachToGrasp`	`Wiggle`

4. **Rearrange.**
A 3D mobile manipulation domain based on ManiSkill-HAB, where a robot rearranges objects to target locations in the presence of large, unpickable obstructions.

- *State and actions:* States are represented using SE(3) poses of all objects. Actions are 10-dimensional joint-position control commands for the mobile manipulator.
- *Object types:* Robot (`r`), pick-and-place objects (`o`), goal locations (`g`), obstruction can (`can`).
- *Operators:* `GoToPickBowl`(?r, ?o), `PickBowl`(?r, ?o), `GoToPlaceBowl`(?r, ?o, ?g), `PlaceBowl`(?r, ?o, ?g), `GoToPickBox`(?r, ?o), `PickBox`(?r, ?o), `GoToPlaceBox`(?r, ?o, ?g), `PlaceBox`(?r, ?o, ?g).
- *Predicates:* `IsBowl`(?o), `IsBox`(?o), `BodyReadyPick`(?r, ?o), `BodyReadyPlace`(?r, ?g), `HoldingBox`(?r), `HoldingBowl`(?r), `HandEmpty`(?r), `AtGoal`(?o, ?g), `BoxAtGoal`(?r), `NoBodyReadyPlace`(?r).
- *Hyperparameters:* Recovery skills are trained with a horizon of 8 and up to $5M$ environment steps at all stages. For concept learning, we collect 500 dreamed trajectories in stage 1.

5. **Cornered Insertion-S.** A UR7e arm with a Robotiq 2F-140 gripper must insert one block onto another to form a tower. One block is initially placed near a box corner, causing the nominal insertion behavior to fail. Train and test tasks differ in goal specification.
   - *State and actions:* States are SE(3) object poses, estimated from wrist-camera scans using SAM3 [45] with depth reprojection and heuristic orientation estimation. Actions are 6-D joint-position commands plus a 1-D gripper command.
   - *Object types:* Robot (`r`), blocks (`b`), box (`box`).
   - *Operators:* `Pick`(?r, ?b), `Insert`(?r, ?b0, ?b1).
   - *Predicates:* `HandEmpty`(?r), `Holding`(?r, ?b), `On`(?b0, ?b1).
   - *Hyperparameters:* Recovery skills follow OmniReset [41], with horizon 200 and up to $5B$ environment steps. Concept learning uses 300 dreamed trajectories in Stage 1.
6. **Cornered Insertion-L.** This domain uses the same hardware setup but adds a lid and a drawer, requiring the robot to place the completed tower into the drawer. Although it shares the recovery skill with Cornered Insertion-S, its task structure leads to different discovered concepts and planning operators.
   - *State and actions:* Same as Cornered Insertion-S.
   - *Object types:* Robot (`r`), blocks (`b`), box (`box`), lid (`lid`), drawer (`drawer`).
   - *Operators:* `Pick`(?r, ?b), `Insert`(?r, ?b0, ?b1), `RemoveLid`(?r, `lid`), `PickTower`(?r, ?b0, ?b1), `PlaceTower`(?r, ?b0, ?b1, `drawer`), `OpenDrawer`(?r, `drawer`), `CloseDrawer`(?r, ?b0, ?b1, `drawer`).
   - *Predicates:* `HandEmpty`(?r), `Holding`(?r, ?b), `On`(?b0, ?b1), `LidOn`(`lid`), `LidOff`(`lid`), `DrawerOpen`(`drawer`), `DrawerClosed`(`drawer`).
   - *Hyperparameters:* Same as Cornered Insertion-S.

E Details about ReSYNC

PDDL Formulation: We use PDDL as the symbolic interface between learned neural abstractions and task-level planning. Each typed object in the RF-MDP is declared as a PDDL object, each initial or learned predicate $\psi \in \Psi$ becomes a typed PDDL predicate, and each skill $\mathtt{C} \in \mathcal{C}$ is represented by a lifted PDDL operator whose parameters match the skill's relational signature. An operator's preconditions specify the lifted predicates that must hold before executing the corresponding ground skill, while its add and delete effects describe the expected symbolic state change after the skill terminates. At planning time, the current continuous state $\mathbf{x}$ is converted into an initial symbolic state by evaluating all predicate classifiers, and the task goal g is provided as a set of goal predicates. The planner searches over ground operators; during execution, each selected operator invokes its associated low-level skill policy, and the symbolic state is re-evaluated after skill termination to determine whether replanning is necessary.

Algorithm 1: Predicate Selection

Input: Candidate predicates $\tilde{\Psi}^{\mathcal{N}}$, dreamed dataset $\mathcal{D}^{\mathcal{N}}$, previous predicates Ψ
Output: Selected predicates $\Psi^{\mathcal{N}}$ and learned operators $\texttt{Op}^{\mathcal{C}^{\mathcal{N}}}$

Initialize $\Psi^{\mathcal{N}} \leftarrow \emptyset$, $J^* \leftarrow \infty$
while *True* **do**
  $\psi^* \leftarrow$ None, $J_{\min} \leftarrow \infty$
  for *each* $\psi \in \tilde{\Psi}^{\mathcal{N}} \setminus \Psi^{\mathcal{N}}$ **do**
    $\hat{\Psi} \leftarrow \Psi \cup \Psi^{\mathcal{N}} \cup \{\psi\}$
    Learn operators $\hat{\texttt{Op}}^{\mathcal{C}^{\mathcal{N}}}$ from $\hat{\Psi}$ and $\mathcal{D}^{\mathcal{N}}$
    Evaluate planning score $J(\hat{\Psi}, \mathcal{D}^{\mathcal{N}})$
    if $J < J_{\min}$ **then**
      $J_{\min} \leftarrow J$, $\psi^* \leftarrow \psi$
  if $J_{\min} \geq J^*$ **then**
    break
  $\Psi^{\mathcal{N}} \leftarrow \Psi^{\mathcal{N}} \cup \{\psi^*\}$
  $J^* \leftarrow J_{\min}$
Learn final operator set $\texttt{Op}^{\mathcal{C}^{\mathcal{N}}}$ from $\Psi \cup \Psi^{\mathcal{N}}$
return $\Psi^{\mathcal{N}}, \texttt{Op}^{\mathcal{C}^{\mathcal{N}}}$

Details about Failure objects: In our experiments, failure objects are identified automatically from simulator interaction signals. We first train an "overfitting" recovery skill on the N user-provided failure states, so that executing the skill makes replanning successful from each of those states. We then roll out this skill and mark as failure objects the objects that experience interaction forces above a fixed threshold. When multiple failure objects are involved, we fit a pairwise Gaussian distribution between each failure object and the relevant object. During both failure mining and dreaming, we resample the states of all failure objects from their corresponding fitted Gaussians.

Terminal Function Learning: At each training stage, we learn two terminal classifiers: one for the failed skill $\texttt{C}^{\mathcal{F}}$ and one for the newly learned recovery skill $\texttt{C}^{\mathcal{N}}$. The terminal classifier $\beta_{\texttt{C}^{\mathcal{F}}}$ is trained by labeling all states preceding the failure state $\mathbf{x}^{\mathcal{F}}$ as `False`, and the failure states $\mathbf{x}^{\mathcal{F}} \sim \mathcal{D}^{\mathcal{F}}$ as `True`. Similarly, the terminal classifier $\beta_{\texttt{C}^{\mathcal{N}}}$ is trained by labeling all states preceding the recovery goal state $\mathbf{x}'$ as `False`, and the goal states $\mathbf{x}'$ as `True`. Both classifiers are implemented as neural networks and optimized using binary cross-entropy loss. The terminal classifier $\beta_{\texttt{C}^{\mathcal{F}}}$ is also used as the failure detector at test time for the RC baseline.

Predicate Pool Learning: Following IVNTR [9], since $\mathcal{D}^{\mathcal{N}}$ is generated through skill execution, we can extract high-level transition tuples of the form $(\mathbf{x}, \underline{\texttt{C}}, \mathbf{x}')$, where $\underline{\texttt{C}}$ denotes a ground skill, and $\mathbf{x}, \mathbf{x}'$ denotes states prior and post the skill execution. For a lifted predicate ψ, the lifted effect vector Δ^{ψ} induces a *ground effect vector* $\mathbf{t}^{\psi,\underline{\texttt{C}}} \in \{-1, 0, 1\}^P$, where P is the number of groundings of ψ over the object set $\mathcal{O}$. Each entry specifies whether a particular ground atom is expected to remain unchanged, be added, or be deleted by the execution of $\underline{\texttt{C}}$. Given a transition $(\mathbf{x}, \underline{\texttt{C}}, \mathbf{x}')$, we apply the predicate classifier to all groundings of ψ:

$$\hat{\mathbf{v}}, \hat{\mathbf{v}}' = \text{Ground}(\mathbf{x}, \theta_\psi),\ \text{Ground}(\mathbf{x}', \theta_\psi) \in [0, 1]^P,$$

obtaining predicted truth values before and after the transition. The effect vector $\mathbf{t}^{\psi,\underline{\texttt{C}}}$ partitions ground atoms into unaffected and affected subsets via masks

$$\mathbf{m}_0 = \mathbb{I}\big(\mathbf{t}^{\psi,\underline{\texttt{C}}} = 0\big), \quad \mathbf{m}^1 = \mathbb{I}\big(|\mathbf{t}^{\psi,\underline{\texttt{C}}}| = 1\big),$$

with $S^1 = \{p \mid m_p^1 = 1\}$ denoting affected atom indexes.

Predicate learning thus enforces consistency between the predicted predicate transitions and the symbolic effects specified by Δ^{ψ}. For unaffected atoms, truth values should remain invariant, for

affected atoms, transitions should match the specified add or delete semantics. Accordingly, we define the per-transition loss

$$\mathcal{L}(\mathbf{x}, \mathbf{x}', \theta_\psi) = \mathcal{L}_{\text{zero}} + \mathcal{L}_{\text{one}},$$

where

$$\mathcal{L}_{\text{zero}} = \text{JS}\Big(\hat{\mathbf{v}} \odot \mathbf{m}_0 \;\Big\|\; \hat{\mathbf{v}}' \odot \mathbf{m}_0\Big)$$

penalizes violations of invariance, and

$$\mathcal{L}_{\text{one}} = \frac{1}{|S^1|} \sum_{p \in S^1} \text{BCE}\Big([\hat{v}_p, \hat{v}'_p], \big[\tfrac{1-t_p}{2}, \tfrac{1+t_p}{2}\big]\Big)$$

penalizes incorrect add ($t_p = 1$) or delete ($t_p = -1$) behavior. Here, $\text{JS}(\cdot\|\cdot)$ denotes Jensen–Shannon divergence and $\text{BCE}(\cdot,\cdot)$ denotes binary cross-entropy. Aggregating across transitions and skills yields the dataset-level objective

$$\mathcal{L}^{\mathcal{D}}(\theta_\psi) = \sum_{\mathtt{C} \in \mathcal{C}^{\mathcal{N}}} \mathbb{E}_{(\mathbf{x}, \underline{\mathtt{C}}, \mathbf{x}') \sim \mathcal{D}_{\mathtt{C}}} \mathcal{L}(\mathbf{x}, \mathbf{x}', \theta_\psi),$$

which can be optimized end-to-end using standard gradient-based methods, without requiring explicit predicate labels. After training with these labels, the classifier for ψ is evaluated on a held-out validation set, yielding a validation loss. Predicates whose classifiers have validation loss below a fixed threshold are retained, forming the final discovered predicate set $\tilde{\Psi}^{\mathcal{N}}$. To constrain the potential space of Δ^ψ, we enumerate the effect profiles with nonzero entries only for a subset of skills, namely the recovery skill $\mathtt{C}^{\mathcal{N}}$, the failed skill $\mathtt{C}^{\mathcal{F}}$, and the skill following recovery.

Predicate Selection: Motivated by previous work [7], ReSYNC explicitly selects a compact subset of predicates that are most useful for planning. Given a set of predicate candidates $\tilde{\Psi}^{\mathcal{N}}$ discovered from dreamed trajectories $\mathcal{D}^{\mathcal{N}}$, ReSYNC evaluates candidate predicate sets using a planning-driven objective $J(\hat{\Psi}, \mathcal{D}^{\mathcal{N}})$. This objective measures the quality of the planning operators derived from $\hat{\Psi}$ along two dimensions: (i) whether the resulting operators can reproduce the skill sequences seen in $\mathcal{D}^{\mathcal{N}}$, and (ii) the efficiency of planning, measured by the number of node expansions required to find the high-level skill-plan. Concretely, for a candidate predicate set $\hat{\Psi}$, we ground predicates over the states in $\mathcal{D}^{\mathcal{N}}$ and learn a corresponding operator set $\hat{\mathtt{Op}}^{\mathcal{C}'}$ following standard symbolic operator learning procedures [22]. We then evaluate $\hat{\mathtt{Op}}^{\mathcal{C}'}$ by planning over the dreamed tasks and computing the resulting score J [7]. To identify an effective and compact abstraction, ReSYNC performs a hill-climbing search over $\tilde{\Psi}^{\mathcal{N}}$. At each iteration, the predicate whose inclusion yields the greatest improvement in the planning objective is added to the selected set. The procedure terminates when no candidate provides further improvement or when a predefined threshold is reached. The final output is the union of the current and the selected predicates $\Psi' = \Psi \cup \Psi^{\mathcal{N}}$ and the corresponding operator set $\mathtt{Op}^{\mathcal{C}'}$, yielding the new abstract planning model. For better understanding, we present an algorithm table for this process in Algorithm 1.

Predicate Finetuning: After stage $(i+1)$, the skill set becomes $\mathcal{C}_{i+1} = \mathcal{C}_i \cup \{\mathtt{C}^{\mathcal{N}}_{i+1}\}$. For each existing predicate $\psi \in \Psi_i$, we extend its effect vector $\Delta^\psi_i \in \{-1, 0, 1\}^M$ to $\Delta^\psi_{i+1} \in \{-1, 0, 1\}^{M+1}$ by computing the effect of the new operators using previous operator learning approach [22]. We then finetune the previously learned predicate classifiers using the updated effect vectors Δ^ψ_{i+1} on the newly dreamed trajectories $\mathcal{D}^{\mathcal{N}}_{i+1}$, following the effect-consistent learning objective introduced above.

F Invariance to Curriculum Order

We further assess the sensitivity of ReSYNC to curriculum design by reversing the sequence of failures ($\mathcal{T}_2 \to \mathcal{T}_1$) in the *Blocked Stacking* domain. As summarized in Table 9, the robot achieves comparable success rates on the final test tasks regardless of the training order. This invariance demonstrates that ReSYNC robustly identifies and integrates relational concepts regardless of the actual experience in the learning curriculum.

Table 9: Robustness to curriculum ordering in *Blocked Stacking*. Reversing the default task sequence ($\mathcal{T}_2 \to \mathcal{T}_1$) yields performance comparable to the original ordering, demonstrating ReSYNC's invariance to the training curriculum.

Task	In		Old		New	
Metric	Acc	Var	Acc	Var	Acc	Var
Default	0.64		0.78		0.83	
Reversed	0.58	-6.0%	0.79	1.0%	0.77	-6.0%

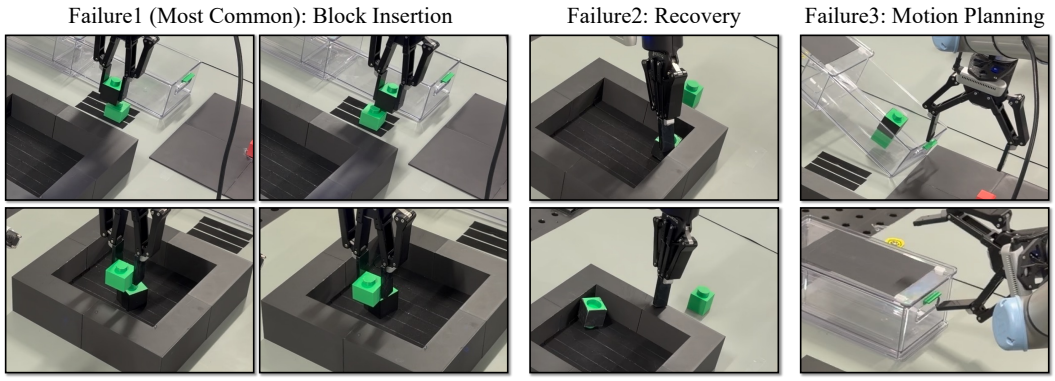

Figure 9: Representative failures in our real-robot experiments.

G Test-time Planning Efficiency

ReSYNC achieves substantially higher planning efficiency than prior work such as DSG-M [5]. As shown in Figure 8, DSG-M learns predicates that primarily characterize terminal outcomes of individual skills. Although such predicates can be useful for specifying local skill order, they provide limited guidance for abstract planning: many potential paths could lead to the goal state. As a result, the planner must explore a larger search space, leading to high planning objectives and multiple redundant high-level plans.

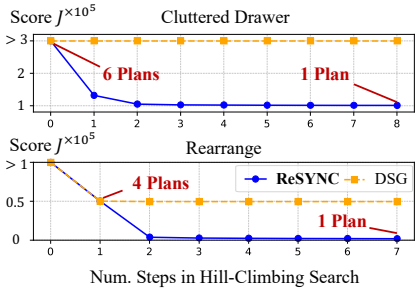

Figure 8: Planning efficiency comparison. ReSYNC learns compact relational concepts that minimize the planning objective and yield single-plan execution, while DSG-M [5] relies solely on terminal predicates, resulting in inefficient planning.

In contrast, ReSYNC explicitly selects predicates according to their impact on the planning objective. The discovered predicates therefore capture not only whether a skill can terminate successfully, but also their contribution to the test-time planning efficiency [7]. This allows ReSYNC to prune spurious symbolic transitions and retain a compact set of meaningful planning choices. Consequently, at test time, ReSYNC consistently produces a single reliable high-level plan, while DSG-M often produces several redundant alternatives that increase planning cost and reduce execution efficiency.

H Noisy Initial Knowledge

While improving the initial knowledge, $\{\mathcal{C}_0, \Psi_0, \mathtt{Op}^{\mathcal{C}_0}\}$, is complementary to our work, we find that ReSYNC is robust to imperfect initialization. In Table 10, we introduced 16%∼38% perceptual noise (w.r.t the mean decision boundary) into the initial predicates. ReSYNC was able to discard unreliable predicates and replace them with newly learned ones, resulting in robust test-time planning performance.

Table 10: Robustness of ReSYNC under noisy initial knowledge. Adding noise to the initial predicate classifiers in Blocked Stacking does not significantly reduce the performance of ReSYNC.

Noise STD	0	0	0.16	0.16	0.38	0.38
Task	$\mathcal{T}_1^{\text{train}}$	$\mathcal{T}_1^{\text{test}}$	$\mathcal{T}_1^{\text{train}}$	$\mathcal{T}_1^{\text{test}}$	$\mathcal{T}_1^{\text{train}}$	$\mathcal{T}_1^{\text{test}}$
Success	0.73	0.76	0.70	0.69	0.66	0.70

I Representative Failures in Real-World Experiments

As shown in Figure 9, we present representative real-robot failure cases. Overall, task planning is reliable, benefiting from the discovered neural predicates and the planning-driven learning objective. The most common failures occur during insertion, where perception noise can misalign the placement pose. This failure mode arises because insertion is executed by a scripted Gaussian-filter-based controller rather than a learned policy, and our system does not use force feedback. The learned RL recovery skill can also fail due to the sim-to-real gap. For example, in the top row, the gripper becomes stuck in the gap, leading to an unrecoverable state; in the bottom row, misaligned image observations cause the gripper to collide with the box. Finally, high-precision skills such as drawer pulling and pushing remain sensitive to perception and control noise. These observations motivate future research on long-horizon planning with dexterous, high-precision, and contact-rich skills, moving beyond primarily prehensile pick-and-place skills [9], where many open problems remain.

J Extended Discussions on the Limitations

Below, we discuss potential approaches for relaxing each assumption in ReSYNC.

The assumption of user provided learning curriculum : One potential approach to automate curriculum design is to allow a coding agent to explore and understand the simulated environment [47]. Specifically, we can prompt a coding agent to attempt planning in various tasks and understand the different failures. Then the coding agent could automate the process of failure-recovery MDP creation and policy learning.

The assumption of being able to identify failure states : This is a common assumption in previous work that learning to recover from failures [3]. There is also orthogonal work on failure detection with foundation models [53–55] that can be combined with our work to enable fully autonomous identification of failure states and subsequent recovery.

The assumption of being able to identify the single relevant object : We follow previous work [10] to derive the failure-relevant object from the failed ground skill. In a more general case where multiple relevant objects are involved, multiple pairwise Gaussian distributions need to be fitted.

The assumption of gaussian distribution : In ReSYNC, the sampled states from the pairwise Gaussians are validated by resetting the physics simulator and stepping it forward for 20 steps. We retain only those states that remain stable under simulation. A promising direction to relax this assumption is to leverage recent diffusion-based generative models for state sampling [8, 56, 57]. For example, we could first learn neural predicates from real trajectory data, and then perform classifier-guided diffusion [49] to generate valid and diverse states for the dreaming phase. This approach would allow us to move beyond simple Gaussian distributions and capture more complex state distributions.

Scalability of compositional dreaming : ReSYNC is designed to operate in batched environments, where both skill policies and predicate classifiers are parameterized by neural networks. This enables efficient parallel execution on GPUs. In practice, even in our most complex domain (Rearrange with 6 objects), the compositional dreaming phase completes in $\sim$2 hours on a single A100 GPU. The primary purpose of compositional dreaming is to enable learning predicate classifiers *from scratch* in a self-supervised manner. This design choice prioritizes generality and minimal prior assumptions, but naturally introduces computational overhead as the number of objects increases. A promising and complementary direction is to leverage pre-trained foundation models to initialize [25] or guide [51] predicate learning. By grounding predicates in pre-trained representations (e.g., from VLMs [51]), and fine-tuning them with limited dreamed data, we can significantly reduce the need for exhaustive compositional dreaming across all object combinations.