SkillWrapper: Generative Predicate Invention for Task-level Robot Planning

Anonymous authors
Paper under double-blind review

Abstract

Generalizing from individual skill executions to long-horizon tasks is a core challenge in building autonomous robots. A promising direction is learning high-level, symbolic representations of low-level robot skills, enabling abstract reasoning independent of the low-level state space. Recent advances in foundation models have made it possible to generate symbolic predicates that operate on raw sensory inputs—a process we call *generative predicate invention*—to facilitate downstream representation learning. However, prior work learns these abstractions using heuristic or ad-hoc procedures, leaving unclear *which* formal properties they ought to satisfy, and *how* these properties can guide representation learning. We address these questions by characterizing conditions under which learned representations support sound and complete task-level planning, and using them to guide the design of SkillWrapper, a system that autonomously learns symbolic representations of black-box skills without predefined tasks, predicates, or operators. Our approach leverages foundation models to actively collect robot data and learn human-interpretable, plannable representations directly from RGB observations. Our extensive empirical evaluation in simulation and on real robots shows that SkillWrapper learns abstract representations that enable robots to compose black-box skills to solve unseen, long-horizon tasks in the real world.

1 Introduction

An autonomous robot operating in the real world must process low-level sensory and motor signals while reasoning about high-level objectives (Doncieux et al., 2018; Konidaris, 2019). Analogous to how humans can perform complex tasks, like cooking or cleaning, without reasoning about muscle-level control, robots should have internal models of their skills that abstract away the intricacies of low-level control. Such models must capture the necessary conditions for executing a skill (e.g., *"pouring from a teapot into a mug requires holding the teapot first"*) and the consequences of doing so (e.g., *"pouring from a teapot into a mug fills the mug"*). These two properties, known as *preconditions* and *effects* in the classical planning literature (Ghallab et al., 2016), enable compositional reasoning to identify long-horizon plans that sequence lower-level skills to solve a task. Typically, these task-level models must be specified manually by a domain expert. However, in real-world settings, such models are nontrivial to acquire due to complex inter-skill constraints specific to the environment and the robot's embodiment. This calls for systems that autonomously learn task-level models of black-box skills, enabling robots to solve long-horizon tasks using off-the-shelf planners.

Previous approaches that learn task-level models for robot planning have taken one of two approaches. Bottom-up methods (Konidaris & Barto, 2009; James et al., 2022; Shah et al., 2025) have provable theoretical properties—i.e., can guarantee that the resulting representations are sound and complete—but either start from restricted, structured states (e.g., configuration space (Shah et al., 2025)) or can learn abstractions from pixel-level inputs but do not exploit foundation models (FMs) and, therefore, may struggle to scale (Konidaris et al., 2018; James et al., 2022). Alternatively, methods driven by FMs exploit their capabilities to generate semantically-meaningful predicates from pixel-level observations—a process we refer to as *generative predicate invention*. On the other hand, while recent work in this line has explored how FMs can be used for predicate invention by generating Python code (Liang et al., 2025) or filling a pool of candidate predicates during hill-climbing search (Athalye et al., 2026), predicate invention is still largely driven by heuristic or ad-hoc procedures. This leaves core questions unanswered: *when* should new predicates be invented, *what* formal properties should

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

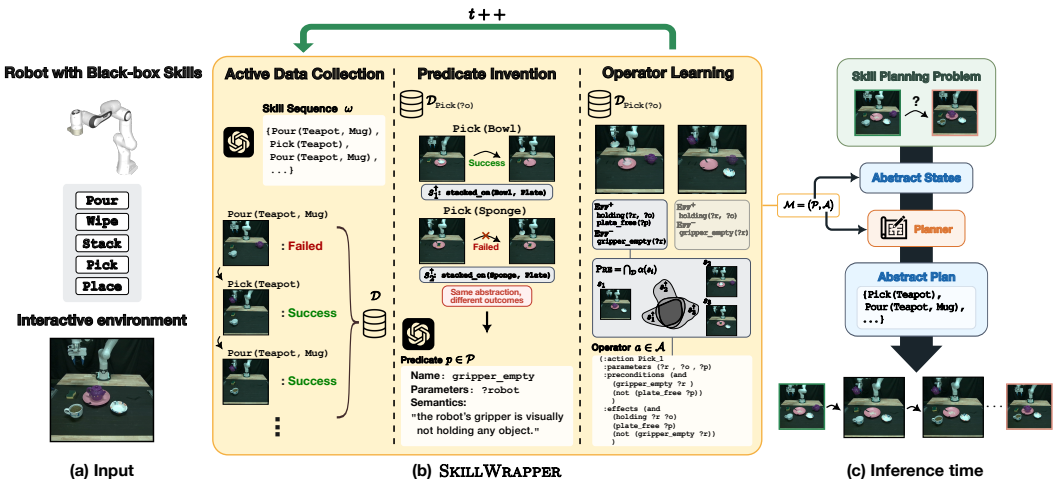


Figure 1: **Overview of SkillWrapper.** For a robot equipped with black-box skills, SkillWrapper learns a task-level model $\mathcal{M}$ that is compatible with off-the-shelf classical AI planners. Given a planning problem specified by low-level initial and goal states, the system computes the corresponding abstract states, then calls a planner to find a solution plan to reach the goal.

the learned predicates and abstract model satisfy, and *how* can these properties guide representation learning?

Our answer is twofold. First, we characterize formal conditions for abstract models to support *sound* and *complete* task-level planning, and use them to derive principled criteria for generative predicate invention. Building on these criteria, we introduce SkillWrapper, a method designed to guide representation learning toward these properties. SkillWrapper uses FMs to interactively collect skill execution data, propose predicates and define their semantics, and classify predicate truth values on low-level observations. Using these data and predicates, SkillWrapper learns human-interpretable symbolic representations of black-box skills that enable task-level planning. To the best of our knowledge, SkillWrapper is the first generative predicate-invention method to learn symbolic representations from scratch without predefined predicates, operators, or tasks for data collection, and the first to use formally characterized model properties to guide predicate invention.

Our core contributions are the following: *1)* SkillWrapper, an autonomous system that leverages FMs to learn human-interpretable symbolic representations of black-box skills without predefined predicates, operators, or tasks for data collection; *2)* a theoretical characterization of conditions under which abstract models support *sound* and *complete* task-level planning, from which we derive principled criteria for predicate invention in SkillWrapper; and *3)* an extensive empirical evaluation of the system in simulation and on real robots.

2 Formal Framework

We consider a problem setting in which a robot is equipped with a set of predefined, "black-box" skills. Because the robot initially lacks a transition model for these skills, it is nontrivial to compose them to solve long-horizon problems. However, if the robot were to acquire a symbolic model of its skills, it could use classical planning to compose them, enabling it to solve unseen tasks. For the reader's convenience, we provide a glossary of notation in Appendix K.

Environment A *domain* is a tuple $\langle \mathcal{S}, \mathsf{T}, \Omega, T \rangle$, where $\mathcal{S}$ is the continuous state space, T is a finite set of object types, and Ω is a library of object-centric skills. The transition function $T : \mathcal{S} \times \Omega \to \mathcal{S}$ characterizes the domain dynamics but is unknown to the robot. We assume deterministic transitions, as is common in related work (Silver et al., 2023; Han et al., 2024; Liang et al., 2025; Athalye et al., 2026); see Appendix E for a full comparison with related problem settings. Within a domain, a *scene* is defined by $\langle s_0, \mathcal{O} \rangle$, consisting of an initial state $s_0 \in \mathcal{S}$ and a set of typed objects $\mathcal{O}$. We denote the type(s) of an object $o \in \mathcal{O}$ as $\mathtt{t}(o) \subseteq \mathsf{T}$.

Black-box object-centric skills We model each skill $\omega \in \Omega$ as a tuple $\omega := \langle \boldsymbol{\theta}_\omega, \mathcal{I}_\omega, \pi_\omega, \beta_\omega \rangle$, where skill parameters $\boldsymbol{\theta}_\omega := \langle \theta^1_\omega, \ldots, \theta^k_\omega \rangle$ specify type constraints $\mathtt{t}(\theta^i_\omega) \in \mathsf{T}$ on object arguments; $\mathcal{I}_\omega : \mathcal{O}^k \to 2^{\mathcal{S}}$ is the *pre-success set*, containing states from which the skill succeeds under the low-level policy π; and $\beta_\omega : \mathcal{O}^k \to 2^{\mathcal{S}}$ is the *success set*, containing states corresponding to successful execution. A skill instance $\underline{\omega} := \omega(\boldsymbol{o})$ is defined for objects $\boldsymbol{o} = \langle o_1, \ldots, o_k \rangle \in \mathcal{O}^k$ iff $\mathtt{t}(\theta^i_\omega) \in \mathtt{t}(o_i)$ for $i \in \{1, \ldots, k\}$. An execution of $\underline{\omega}$ from state $s \in \mathcal{S}$ produces a transition $\langle s, \underline{\omega}, s' \rangle$, where $s' = T(s, \underline{\omega})$. We consider skill executions initiated outside the corresponding success set, excluding attempts whose objective is already satisfied. An execution succeeds if and only if it starts from the pre-success set ($s \in \mathcal{I}_{\underline{\omega}}$), in which case it terminates in the success set ($s' \in \beta_{\underline{\omega}}$). We allow a type-compatible skill instance to be attempted outside $\mathcal{I}_{\underline{\omega}}$, in which case the execution fails. We assume that the robot can observe whether an execution succeeds or fails, but does not know $\mathcal{I}_{\underline{\omega}}$ or $\beta_{\underline{\omega}}$ a priori, so the skills are considered black boxes.

Symbolic abstractions and groundings To enable reasoning over a discrete space of possible abstract states, a set of predicates is denoted $p \in \mathcal{P}$, each defined by $p := \langle \ell_p, \boldsymbol{\theta}_p, C_p \rangle$, where ℓ_p is a semantic label, and parameters $\boldsymbol{\theta}_p$ define type constraints on object arguments of the classifier C_p. Grounding a predicate p with type-compatible objects induces a grounded predicate $\underline{p}$ and its classifier $C_{\underline{p}} : \mathcal{S} \to \{0, 1\}$, which tests whether the grounded predicate holds in a given state. Given a set of predicates $\mathcal{P}$ and a set of objects $\mathcal{O}$, let $\underline{\mathcal{P}}$ denote the set of all possible grounded predicates, and define the *abstract state space* as $\mathcal{S}^\uparrow := 2^{\underline{\mathcal{P}}}$, the symbolic counterpart to the state space $\mathcal{S}$. The *abstraction function* $\alpha : \mathcal{S} \to \mathcal{S}^\uparrow$ maps each low-level state to the set of grounded predicates true in that state (i.e., the corresponding abstract state), while the *grounding function* $\Gamma : \mathcal{S}^\uparrow \to 2^{\mathcal{S}}$ maps an abstract state $s^\uparrow \in \mathcal{S}^\uparrow$ to the low-level states in which its grounded predicates hold.

An operator $a \in \mathcal{A}$ is defined by $a := \langle \omega_a, \boldsymbol{\theta}_a, \textsc{Pre}_a, \textsc{Eff}^+_a, \textsc{Eff}^-_a \rangle$, denoting a skill ω_a, typed parameters $\boldsymbol{\theta}_a$, preconditions $\textsc{Pre}_a \subseteq \mathcal{P}$, add effects $\textsc{Eff}^+_a \subseteq \mathcal{P}$, and delete effects $\textsc{Eff}^-_a \subseteq \mathcal{P}$. Given operators $\mathcal{A}$ and objects $\mathcal{O}$, the *abstract action space* $\underline{\mathcal{A}}$ is the set of all valid operator groundings $\underline{a} = a(\boldsymbol{o})$, defined for objects $\boldsymbol{o} = \langle o_1, \ldots, o_m \rangle \in \mathcal{O}^m$ iff $\mathtt{t}(\theta^i_a) \in \mathtt{t}(o_i)$ for $i \in \{1, \ldots, m\}$; each abstract action $\underline{a}$ defines ground preconditions $\underline{\textsc{Pre}}_{\underline{a}} \subseteq \underline{\mathcal{P}}$, ground add effects $\underline{\textsc{Eff}}^+_{\underline{a}} \subseteq \underline{\mathcal{P}}$, and ground delete effects $\underline{\textsc{Eff}}^-_{\underline{a}} \subseteq \underline{\mathcal{P}}$. We refer to the tuple $\mathcal{M} := \langle \mathcal{P}, \mathcal{A} \rangle$ as an *abstract transition model*, and an *abstract plan* found using $\mathcal{M}$ is a sequence of abstract actions: $\underline{\boldsymbol{a}} := [\underline{a}_1, \ldots, \underline{a}_n]$.

Planning problems A *skill planning problem* is defined by $\langle \mathcal{O}, s_0, \mathcal{S}_g \rangle$, which is a set of objects $\mathcal{O}$, an initial state $s_0 \in \mathcal{S}$, and goal states $\mathcal{S}_g \subseteq \mathcal{S}$. A solution to a skill planning problem is any *skill plan* $\underline{\boldsymbol{\omega}} = [\underline{\omega}_1, \ldots, \underline{\omega}_k]$ such that for $i \in \{1, \ldots, k\}$, $s_i = T(s_{i-1}, \underline{\omega}_i)$, $s_{i-1} \in \mathcal{I}_{\underline{\omega}_i}$, and $s_k \in \mathcal{S}_g$.

Problem statement Given a domain $\langle \mathcal{S}, \mathsf{T}, \Omega, T \rangle$ containing scenes $\{\langle s_0, \mathcal{O} \rangle\}_{i=1}^N$, we define the problem as learning a vocabulary and an abstract transition model $\mathcal{M} := \langle \mathcal{P}, \mathcal{A} \rangle$ for the skills Ω defined over T, such that when given a skill planning problem $\langle \mathcal{O}, s_0, \mathcal{S}_g \rangle$ in a novel scene, a complete symbolic planner using $\mathcal{M}$ will find a valid abstract plan $\underline{\boldsymbol{a}}$, whenever a feasible skill plan exists.

3 Method

We introduce SkillWrapper, a system that autonomously learns representations for black-box skills. The algorithm iterates through three steps: *1)* active exploration for data collection, *2)* predicate invention based on contrastive examples, and *3)* operator learning using the accumulated data and predicates. As SkillWrapper continues to collect data, invent and incorporate new predicates, and update its planning model, it progressively improves the fidelity of these learned representations. Full pseudocode is in Appendix A.

3.1 Active Data Collection

To collect data for learning, our method selects a sequence of skill instances $\underline{\boldsymbol{\omega}} = [\underline{\omega}_1, \ldots, \underline{\omega}_k]$ for the robot to execute. After executing each $\underline{\omega}_i$, the robot records a *transition* $\langle s_{i-1}, \underline{\omega}_i, s_i \rangle$ capturing both the *executability* of the skill instance (i.e., whether $s_{i-1} \in \mathcal{I}_{\underline{\omega}_i}$) and its *outcome* (i.e., the resulting state s_i). These transitions are accumulated into a dataset $\mathcal{D} = \{\langle s, \underline{\omega}, s' \rangle\}$ used for predicate invention and operator learning. We define $\mathcal{D}_\omega$ as the set of transitions in $\mathcal{D}$ involving a skill $\omega \in \Omega$. To propose a skill sequence, SkillWrapper first prompts an FM with the robot skills Ω and the current predicates $\mathcal{P}$, which may be empty, to generate a candidate pool. Each candidate sequence

is scored along two axes, *balance* (b) and *coverage* (c). SkillWrapper retains the Pareto-optimal subset containing all sequences i for which no other sequence j has $b_j \geq b_i$ and $c_j \geq c_i$. The final skill sequence is sampled uniformly from this subset.

Balance (b) encourages candidate skill sequences to contain a desired mixture of successful and failed executions. To estimate this mixture, SkillWrapper simulates each candidate sequence using the current abstract model, maintaining a predicted abstract state as the sequence progresses. At each step t, a skill instance $\underline{\omega}_t$ is predicted to succeed if at least one associated ground operator $\underline{a}_t \in \underline{\mathcal{A}}_{\underline{\omega}_t}$ is applicable, i.e., its preconditions satisfy $s^{\uparrow}_{t-1} \models \underline{\textsc{Pre}}_{\underline{a}_t}$. If multiple operators are applicable, we select the most specific one, defined as the operator with the largest number of preconditions, so that conditional effects are applied only when their corresponding conditions are satisfied. The abstract state is then updated according to the selected operator, $s^{\uparrow}_t \leftarrow (s^{\uparrow}_{t-1} \setminus \underline{\textsc{Eff}}^-_{\underline{a}_t}) \cup \underline{\textsc{Eff}}^+_{\underline{a}_t}$; if no operator is applicable, the skill is predicted to fail and the abstract state remains unchanged. Let $\hat{r}_{\text{suc}}$ denote the fraction of skill executions predicted to succeed over the simulated sequence. Given a target success ratio r^*_{suc}, the balance score is $b = -|\hat{r}_{\text{suc}} - r^*_{\text{suc}}|$. A balanced target (e.g., $r^*_{\text{suc}} = 0.5$) encourages both successful and failed executions, providing contrastive examples for predicate invention, while a higher target (e.g., $r^*_{\text{suc}} = 0.9$) favors successful executions and provides more data for learning skill effects.

Coverage (c) encourages candidate sequences to cover pairs of consecutively executed skills more uniformly, helping SkillWrapper expose interdependencies between skills' preconditions and effects. We maintain a skill-bigram count matrix Q, where each entry $Q[i, j]$ records how often the consecutive pair $(\underline{\omega}_i, \underline{\omega}_j)$ has appeared in the collected dataset $\mathcal{D}$. For a candidate sequence $\underline{\boldsymbol{\omega}}$, let Q' denote the hypothetical count matrix after additionally counting the consecutive skill pairs in $\underline{\boldsymbol{\omega}}$. We score the candidate by how much it increases the entropy of this empirical bigram distribution, $c(\underline{\boldsymbol{\omega}}) = H(Q') - H(Q)$, where $H(Q) = -\sum_{i,j} p_{i,j} \log p_{i,j}$ and $p_{i,j} = Q[i, j]/\sum_{k,\ell} Q[k, \ell]$ is the normalized frequency of each skill pair. Thus, sequences containing relatively underexplored skill pairs tend to receive higher coverage scores.

3.2 Predicate Invention

Unlike prior work that requires an initial set of predicates (e.g., goal predicates) (Silver et al., 2023; Liang et al., 2025; 2026; Athalye et al., 2026) or partial operators (e.g., with pre-specified effects) (Liang et al., 2025; 2026), SkillWrapper begins without predicates or operators ($\mathcal{P} = \mathcal{A} = \emptyset$), instead bootstrapping based on black-box skill executions.

Given grounded predicates $\underline{\mathcal{P}}$ and an object tuple $\boldsymbol{o} \in \mathcal{O}^k$, we define a *lifting function* $\Lambda_{\boldsymbol{o}}$ that replaces the objects in $\boldsymbol{o}$ with typed variables, allowing predicates from different skill instances to be compared in a common lifted form (details in App. F). We also define the *abstract change* between states $s, s' \in \mathcal{S}$ as $\Delta(s, s') = \langle \alpha(s') \setminus \alpha(s), \alpha(s) \setminus \alpha(s') \rangle$.

Conditions for predicate invention. SkillWrapper invents new predicates only when the task-level model cannot explain newly observed transitions. To formalize when the current vocabulary is insufficient, for each skill instance $\underline{\omega}$, we define the set of states from which the model predicts $\underline{\omega}$ would succeed and in which it predicts that $\underline{\omega}$ has succeeded:

$$\Phi_{\underline{\omega}} \coloneqq \bigcup_{\underline{a} \in \underline{\mathcal{A}}_{\underline{\omega}}} \Gamma(\underline{\textsc{Pre}}_{\underline{a}}), \quad \Xi_{\underline{\omega}} \coloneqq \bigcup_{\underline{a} \in \underline{\mathcal{A}}_{\underline{\omega}}} \Gamma((\underline{\textsc{Pre}}_{\underline{a}} \setminus \underline{\textsc{Eff}}^-_{\underline{a}}) \cup \underline{\textsc{Eff}}^+_{\underline{a}}),$$

where $\underline{\mathcal{A}}_{\underline{\omega}}$ denotes the abstract actions for $\underline{\omega}$.

The first condition arises when the learned model incorrectly predicts the pre-success set $\mathcal{I}_\omega$ of a skill. Concretely, this occurs when two transitions involving instances of the same skill, one successful and one failed, are both predicted to succeed by the current model. Letting $\underline{\omega}_i = \omega(\boldsymbol{o}_i)$ and $\underline{\omega}_j = \omega(\boldsymbol{o}_j)$:

$$\exists \langle s_i, \underline{\omega}_i, s'_i \rangle, \langle s_j, \underline{\omega}_j, s'_j \rangle \in \mathcal{D}_\omega \text{ s.t. } s_i \in \mathcal{I}_{\underline{\omega}_i} \text{ and } s_j \notin \mathcal{I}_{\underline{\omega}_j}, \text{but } s_i \in \Phi_{\underline{\omega}_i}, s_j \in \Phi_{\underline{\omega}_j}. \tag{1}$$

The second condition detects when the predicate vocabulary cannot distinguish between the effects of a successful and a failed execution of the same skill. We use the lifting function $\Lambda_{\boldsymbol{o}}$ to account for the fact that the two transitions may involve different object arguments (e.g., $\boldsymbol{o}_i$ and $\boldsymbol{o}_j$):

$$\begin{aligned} &\exists \langle s_i, \omega(\boldsymbol{o}_i), s'_i \rangle, \langle s_j, \omega(\boldsymbol{o}_j), s'_j \rangle \in \mathcal{D}_\omega \text{ s.t. } s'_i \in \beta_{\omega(\boldsymbol{o}_i)} \text{ and } s'_j \notin \beta_{\omega(\boldsymbol{o}_j)}, \text{ yet} \\ &\Lambda_{\boldsymbol{o}_i}(\Delta(s_i, s'_i)) = \Lambda_{\boldsymbol{o}_j}(\Delta(s_j, s'_j)). \end{aligned} \tag{2}$$

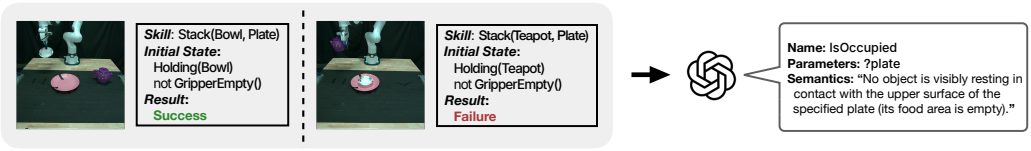


Figure 2: **Example of Predicate Invention.** The initial states of the two transitions both satisfy the preconditions of certain operators of the `Stack` skill, while one transition is successful, but the other is not. In this case, condition 1 is triggered, and the FM is prompted to generate a new predicate.

Predicate implementation. We employ FMs to generate and evaluate predicates: *1) Predicate generation*: Given a contrastive pair of transitions as identified by Conditions 1 or 2, SkillWrapper prompts the FM to generate a candidate predicate $\hat{p}$ to distinguish the pair defined by the semantic label ℓ_p (see Fig. 2). *2) Predicate evaluation*: Given a grounded predicate $\underline{p}$ and an image-based low-level state $s \in \mathcal{S}$, the FM is queried for the truth value of $\underline{p}$ in s by its semantic label ℓ_p.

Predicate selection. We introduce a scoring function to estimate the utility of a candidate predicate $\hat{p}$ by learning hypothetical operators $\hat{\mathcal{A}}$ under the vocabulary $\hat{\mathcal{P}} = \mathcal{P} \cup \{\hat{p}\}$. For each successful transition $\langle s_i, \underline{\omega}_i, s_i' \rangle \in \mathcal{D}$, there must exist at least one ground operator $\underline{\hat{a}} \in \underline{\hat{\mathcal{A}}}$ with $s_i \models \underline{\textsc{Pre}_{\hat{a}}}$; for each failed transition $\langle s_j, \underline{\omega}_j, s_j' \rangle \in \mathcal{D}$, no such operator may exist. Effect scoring follows the same principle based on the abstract change $\Delta(s, s')$. After evaluating over $\mathcal{D}$, a candidate predicate is kept if its score exceeds an empirical threshold. Thus, whether a generated predicate is added to the vocabulary is determined using all observed transitions, while the contrastive pair provides visual cues to the FM for proposing a candidate. Further details can be found in Algorithm 6 (App. A).

Empirical noise handling. FM-generated predicates and their evaluated truth values can be imperfect due to real-world noise. SkillWrapper handles such noise empirically in two places: during predicate selection, each candidate is scored over the full set of observed transitions, making the selection robust to a small fraction of misclassifications and filtering occasional low-quality predicates; during operator learning, small effect clusters are pruned as noisy outliers, following standard practice in symbolic model learning. Together, these mechanisms reduce the impact of occasional FM errors without assuming oracle predicate generation or classification. Our experiments demonstrate that SkillWrapper remains effective under real-world noise, while the component-wise evaluation in Appendix H provides additional evidence of FM reliability.

3.3 Operator Learning

A single skill may produce distinct abstract state changes depending on the context of execution. For example, `GoTo(desk)` may bring objects on a desk into reach, if the desk is not empty. To represent these *conditional effects*, we extend the *associative model learning* paradigm (Arora et al., 2018) to our setting. SkillWrapper clusters observed transitions based on their lifted abstract changes $\Lambda_{\boldsymbol{o}}(\Delta(s, s'))$, enabling it to learn a single operator across distinct instantiations $\omega(\boldsymbol{o})$ of a skill ω (cf. Eq. 2). For each cluster $\boldsymbol{d} \subseteq \mathcal{D}_\omega$, we obtain its preconditions by intersecting the lifted initial abstract states of all transitions in the cluster: $\textsc{Pre} = \bigcap_{i;\, \tau_i \in \boldsymbol{d}} \Lambda_{\boldsymbol{o}_i}(s_i^{\uparrow})$, where $\tau_i = \langle s_i^{\uparrow}, \omega(\boldsymbol{o}_i), s_i'^{\uparrow} \rangle$. We retain only predicates involving at least one of the skill arguments in $\boldsymbol{o}_i$.

In realistic domains, objects might belong to multiple overlapping categories (e.g., a `Mug` is `fillable`, while a `Bottle` is both `fillable` and `openable`). To solve this, SkillWrapper conservatively assigns operator parameters with the lowest level of the type hierarchy consistent with the data, preventing over-generalization. As a result, the learned predicate set remains compact, informative, and aligned with the most recent transition data. See Appendix F for a step-by-step example.

3.4 Theoretical Analysis

We first characterize correctness, soundness, and completeness of an abstract transition model, then establish empirical consistency and asymptotic convergence properties of SkillWrapper. Full proofs of the theoretical results are provided in Appendix C.

Model properties. We consider three properties of an abstract model $\mathcal{M}$. A model is *correct* if its predicted pre-success Φ_ω and success sets Ξ_ω exactly match those of every skill; *soundness*

requires every abstract plan to correctly predict the outcome of its corresponding skill execution, while *completeness* requires every feasible skill plan to have a corresponding abstract plan. We provide the formal definitions in Appendix B.

SkillWrapper can learn conditional effects by representing distinct effects of the same skill with separate operators. When the preconditions of such operators overlap, however, multiple operators may be simultaneously applicable while predicting different outcomes, making a universal soundness guarantee inherently difficult without additional disambiguation. We therefore define soundness over effect-unambiguous plans, while still allowing SkillWrapper to learn and use conditional effect operators in practice.

Proposition 1. *Any correct model learned by* SkillWrapper *is sound and complete.*

Proof sketch. Correctness ensures that the model exactly captures where each skill succeeds and its successful outcomes. By construction, SkillWrapper clusters successful transitions by effect and derives each operator's preconditions from the same cluster, preserving their correspondence; effect-unambiguity then ensures an applicable operator predicts the corresponding abstract effect. Applying this correspondence inductively over a skill sequence establishes soundness and completeness. □

Next, we analyze how SkillWrapper approaches this property through its predicate invention conditions. Lemma 1 shows that, after all identified contrastive transition pairs have been resolved, the estimated and true success sets agree on the observed terminal states. Together with operator learning and the precondition invention condition, this yields Theorem 1, which establishes empirical consistency of both the pre-success and success sets on $\mathcal{D}$. This result forms the basis for the asymptotic convergence property in Theorem 2.

Lemma 1. *Let $\mathcal{M}$ be the model learned by* SkillWrapper *from dataset $\mathcal{D}$ after all contrastive transition pairs satisfying the predicate-invention conditions have been resolved. The estimated success set agrees with the true success set on every observed terminal state: for each transition $\langle s, \underline{\omega}, s' \rangle \in \mathcal{D}, s' \in \Xi_{\underline{\omega}} \iff s' \in \beta_{\underline{\omega}}$.*

Proof Sketch. Suppose not. Then some observed transition $\tau = \langle s, \underline{\omega}, s' \rangle$ has inconsistent terminal membership. If $s' \in \Xi_{\underline{\omega}} \setminus \beta_{\underline{\omega}}$, then τ is failed but s' satisfies the conditions predicted by a learned operator. Thus, τ has the same lifted abstract effect as a successful transition supporting that operator, triggering Condition 2, a contradiction. Conversely, if $s' \in \beta_{\underline{\omega}} \setminus \Xi_{\underline{\omega}}$, then τ is successful under our definition. Operator learning therefore constructs an operator covering τ, implying $s' \in \Xi_{\underline{\omega}}$, again a contradiction. □

Theorem 1 (Empirical Consistency of SkillWrapper)**.** *Let $\mathcal{M}$ be the model learned by* SkillWrapper *from dataset $\mathcal{D}$ after all identified contrastive transition pairs have been resolved. The estimated pre-success and success sets agree with their true counterparts on every observed transition $\langle s, \underline{\omega}, s' \rangle \in \mathcal{D}$: $s \in \Phi_{\underline{\omega}} \iff s \in \mathcal{I}_{\underline{\omega}}$, $s' \in \Xi_{\underline{\omega}} \iff s' \in \beta_{\underline{\omega}}$.*

Proof Sketch. For each observed transition $\langle s, \underline{\omega}, s' \rangle \in \mathcal{D}$, if $s \in \mathcal{I}_{\underline{\omega}}$, the transition is successful and belongs to a cluster used for operator learning. By construction, the operator learned from this cluster is applicable at s, so $s \in \Phi_{\underline{\omega}}$. Conversely, if $s \in \Phi_{\underline{\omega}} \setminus \mathcal{I}_{\underline{\omega}}$, this failed transition and a successful transition from which the applicable operator was learned would trigger Condition 1, contradicting the resolution of all contrastive pairs. Thus, the pre-success memberships agree. The success-set equivalence follows directly from Lemma 1. □

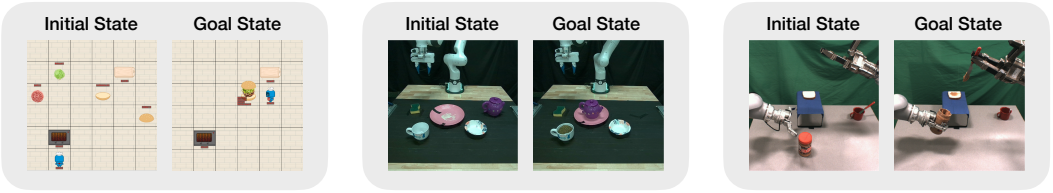
To account for temporal dependence in collected data, let $\tau_{r,t}$ denote transition t in episode r and $\mathcal{D}_{1:r-1}$ the preceding collection history. Conditional on this history, we model each episode's transitions as a possibly nonstationary β-mixing process, with dependence between observations separated by k steps bounded by $\beta(k)$ (Yu, 1994). For a candidate model $\mathcal{M}$, define the transition-level error and its average conditional error across K episodes as

$$\ell_{\mathcal{M}}(\tau) \coloneqq \mathbf{1}\{ s \in \mathcal{I}_{\underline{\omega}} \ominus \Phi_{\underline{\omega}} \ \vee \ s' \in \beta_{\underline{\omega}} \ominus \Xi_{\underline{\omega}} \},$$

$$d_{\mathrm{compl},K}(\mathcal{M}, \mathcal{M}^*) \coloneqq \frac{1}{K} \sum_{r=1}^{K} \mathbb{E}\left[\frac{1}{n} \sum_{t=1}^{n} \ell_{\mathcal{M}}(\tau_{r,t}) \,\middle|\, \mathcal{D}_{1:r-1} \right].$$

where $\ominus$ denotes symmetric difference. We first establish the corresponding single-episode bound in Lemma 2 (Appendix C), and use it below to characterize convergence across episodes.

Theorem 2 (Asymptotic Convergence of SkillWrapper)**.** *Let $\mathcal{H}$ be a fixed finite model hypothesis class, and let $\widehat{\mathcal{M}}_K \in \mathcal{H}$ satisfy Theorem 1 on all transitions from K episodes of length n. For any*



(a) Robotouille (b) Panda (c) Bimanual Kuka

Figure 3: Skill Planning Problem in Different Domains.

$\epsilon \in (0,1)$ and integer $1 \leq k \leq n$, let $m = \lfloor n/k \rfloor$. With probability at least $1 - |\mathcal{H}|(e^{-Km\epsilon} + K(m-1)\beta(k))$, the learned model $\widehat{\mathcal{M}}_K$ satisfies $d_{\text{compl},K}(\widehat{\mathcal{M}}_K, \mathcal{M}^*) \leq \epsilon$. Moreover, this error converges to zero in probability as the number of episodes increases.

Proof Sketch. Conditioning on previously collected data, we apply the single-episode bound of Lemma 2 within each episode. Iterated conditional expectation combines the evidence across episodes, bounding the probability that a fixed model remains empirically consistent while its average conditional error exceeds ϵ by $e^{-Km\epsilon} + K(m-1)\beta(k)$. A union bound over $\mathcal{H}$, together with Theorem 1, yields the stated guarantee. Choosing $k = n$ eliminates the mixing penalty and makes the failure bound vanish as $K \to \infty$, establishing convergence. □

It is important to note that Lemma 1 and Theorem 1 are empirical consistency results and do not require globally correct FM predictions; they establish consistency with the observed transition labels. Theorem 2, together with its single-episode result in Lemma 2, additionally assumes correct FM-provided predicate labels. While real-world labels are inevitably noisy, our experiments show that the theoretically derived conditions remain effective in practice.

4 Experiments

We conduct simulated and real-world experiments to evaluate: 1) the *effectiveness* of the planning abstractions learned by SkillWrapper on unseen problems, 2) how well these abstractions *generalize* across environments, and 3) the *data efficiency* of learning from skill executions. All experiments operate on image observations, and use GPT-5 (OpenAI, 2025) for predicate generation and evaluation. The most closely related generative predicate invention methods (Liang et al., 2025; Athalye et al., 2026; Liang et al., 2026) operate in a "completing partial operators" setting, assuming additional supervision, such as predefined predicates, partial operators, or training tasks, and are therefore not directly comparable to our "learning from scratch" setting (details discussed in Appendix E). We therefore use applicable baselines and controlled ablations, with *FM Invent* serving as the closest adaptation of Pix2Pred (Athalye et al., 2026) to our setting. Our evaluation spans six scenes across a simulated domain and two physical robot platforms. Quantitative results are averaged over five independent runs for simulated experiments, and three for real-robot experiments. In each run, SkillWrapper independently collects data, invents predicates, and learns the abstract model. In addition to the full-system results discussed in this section, we provide a comprehensive evaluation of the FM components of SkillWrapper in Appendix H, including predicate invention comparisons, predicate classification accuracy, unseen objects, and varied lighting conditions or viewing angles.

4.1 Simulated Experiments

We conduct experiments in Robotouille (Gonzalez-Pumariega et al., 2025), a simulated kitchen environment with six objects and a robot with five skills. We design and categorize 50 skill planning problems for this domain: 20 easy problems, whose optimal solutions have at most 7 steps; 20 hard problems, whose optimal solutions have at most 15 steps; and 10 impossible tasks that cannot be realized in the environment, which a sound planning model must identify. We compare SkillWrapper against the following baselines:

- **Expert Operators** are manually defined by a human expert as PDDL predicates and operators.
- **System Predicates** uses the simulator's built-in predicates, which are designed to unambiguously define any simulated state, and knows the ground-truth abstract state; cannot invent new predicates.
- **ViLa** (Hu et al., 2023) iteratively prompts an FM for the next action, given the current image observation and the robot's action history, until the goal state is reached.

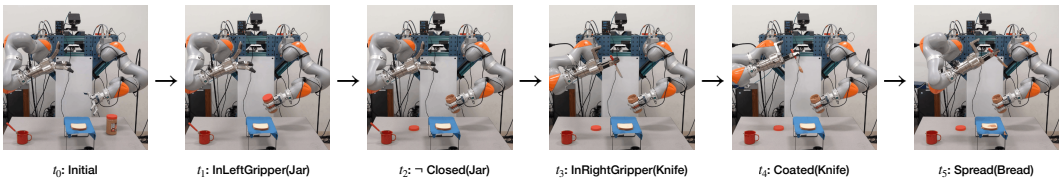


Figure 4: Example predicate truth values during a sequence of bimanual robot skill executions.

- **Random Exploration** ablates the data collection strategy of SkillWrapper by randomly sampling skills and valid object arguments. Predicate invention and operator learning are unchanged.
- **FM Invent** ablates the predicate invention method of SkillWrapper by directly prompting a foundation model (FM) with all image states to invent predicates in batches. The active data collection and operator learning procedures are the same as SkillWrapper.
- **No Heuristic** is the same as SkillWrapper, except that skill sequences are selected randomly from the foundation model's output without applying the balance and coverage heuristics (Sec. 3.1).

We evaluate each method on the problem sets and report the average results in Table 1, where *Solved %* is the percentage of problems that were successfully solved, or the proportion of impossible tasks that were correctly identified by returning an empty plan, and *planning budget* (PB) is defined as the number of plans attempted before solving a problem (adopted from Liang et al. (2025)). Impossible problems probe soundness, since a *sound* model cannot admit spurious abstract plans for goals that are infeasible in the environment. We find that SkillWrapper outperforms all baselines without access to privileged knowledge, and even surpasses the performance of the System Predicates baseline on hard problems. Due to space constraints, we defer case studies, failure modes, and additional analysis to Appendix G.

4.2 Robot Experiments

To evaluate the applicability of SkillWrapper for real-world settings, we conduct two sets of experiments on two robotic platforms: a tabletop Franka Emika Panda and a bimanual manipulator with two Kuka iiwa arms. These experiments investigate compositional generalization and data efficiency, respectively. See our supplementary materials for video demonstrations of both experiments, and Appendix I.4 for further details about our experimental setup.

Generalization In this setting, a tabletop Panda manipulator has a skill set Ω consisting of five skills, and the object set $\mathcal{O}$ contains five objects (see Fig. 1(a)). To evaluate the compositional generalization ability of the learned planning abstractions, we design three smaller training environments, each containing a subset of $\mathcal{O}$, and thus only a subset of Ω is executable. Each training environment contains fewer than 10 possible abstract states under the expert-defined model. After running SkillWrapper for one iteration in each environment, we port the learned model to a test environment containing all objects in $\mathcal{O}$, inducing 34 possible abstract states. We prepare an evaluation set consisting of five problems in the training environments, five problems in the test environment, and five *impossible* problems. We report the results of this experiment in Table 2.

Data efficiency In this setting, a bimanual manipulator robot is equipped with six skills in an environment containing three objects. Notably, this environment contains multiple dead ends, hindering data collection. This design is intended to test the data efficiency of SkillWrapper over multiple learning iterations. Figure 4 depicts an example sequence of predicate truth value changes induced by a skill sequence. As shown in Figure 5, SkillWrapper improves in performance as it collects additional data and invents new predicates, surpassing all baselines within five iterations.

Method	Easy		Hard		Impossible
	Solved % ↑	PB ↓	Solved % ↑	PB ↓	
Expert Ops.	**81.0 ± 3.7**	**1.9 ± 0.4**	**58.1 ± 3.9**	**4.2 ± 0.4**	**100 ± 0**
Sys Preds.	**79.0 ± 3.7**	2.6 ± 0.2	22.0 ± 12.9	7.8 ± 1.3	42.0 ± 7.5
ViLa	46.0 ± 16.2	-	13.9 ± 11.6	-	20.0 ± 10.9
Random	4.0 ± 2.0	9.6 ± 0.2	0 ± 0	10.0 ± 0	100 ± 0
FM Invent	23.3 ± 16.5	7.7 ± 1.7	1.7 ± 2.4	9.7 ± 0.2	63.3 ± 26.2
No Heuristic	**76.0 ± 4.9**	**2.5 ± 0.9**	24.0 ± 19.6	7.8 ± 1.8	80 ± 20.9
Ours	**74.0 ± 3.7**	**2.7 ± 0.4**	**40.0 ± 3.2**	**6.3 ± 0.4**	**100 ± 0**

Table 1: Results in Robotouille Environment

Method	In-domain		Generalization		Impossible
	Solved % ↑	PB ↓	Solved % ↑	PB ↓	
Expert Ops.	66.7 ± 9.4	3.3 ± 0.9	53.3 ± 9.4	5.3 ± 0.9	46.7 ± 0
ViLa	46.7 ± 9.4	-	6.7 ± 9.4	-	6.7 ± 9.4
Random	0 ± 0	10.0 ± 0	0 ± 0	10.0 ± 0	60 ± 16.3
FM Invent	26.7 ± 18.9	7.3 ± 1.9	13.3 ± 9.4	8.7 ± 0.9	86.7 ± 9.4
No Heuristic	53.3 ± 9.4	4.7 ± 0.9	40.0 ± 16.3	6.0 ± 1.6	53.3 ± 9.4
Ours	**76.7 ± 9.4**	**2.7 ± 0.9**	**60.0 ± 0**	**4.0 ± 0**	**66.7 ± 9.4**

Table 2: Results of Generalization Experiment

Discussion SkillWrapper performs competitively with the Expert Operators while requiring only a handful of exploratory interactions, and generalizes well, suggesting that the invented predicates and learned model capture meaningful abstractions rather than overfitting. The gap between SkillWrapper and ViLa demonstrates the benefit of an explicit symbolic model over naive FM prompting; the gap against Random Exploration and No Heuristic shows the importance of guided data collection; and the gap against FM Invent shows that our predicate invention algorithm, not the FM alone, enables efficient model learning. Together, these results indicate that SkillWrapper achieves a favorable combination of data efficiency and model fidelity. Further analysis is in Appendix G.

Qualitatively, we observe that predicate invention is primarily driven by preconditions (Eq. 1), because any individual skill effect is typically necessary to represent only when another skill's precondition depends on it. Effect-driven predicate invention (Eq. 2) is the necessary secondary mechanism for terminal skills without downstream dependents; without it, tasks like spreading peanut butter (upon which, in our setting, no other skill depends) could not be modeled.

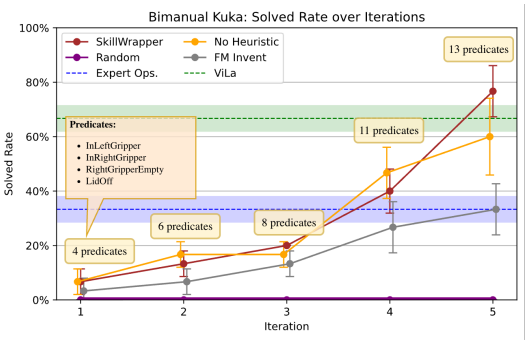


Figure 5: Bimanual manipulation results ($n = 3$).

These results show that SkillWrapper is effective on real robots: planning models learned in restricted domains generalize to larger problems, and can be progressively improved even in settings with irreversible actions. By outperforming all non-privileged baselines, and performing competitively against hand-specified or privileged baselines, SkillWrapper demonstrates the importance of predicate invention and active learning in enabling symbolic representations to scale to embodied tasks.

5 Related Work

SkillWrapper draws ideas from different fields of research, such as model learning, abstraction learning, and task and motion planning (TAMP). Several methods have used FMs as high-level planners (Ahn et al., 2022; Rana et al., 2023; Driess et al., 2023). Recent work has also explored how representations can be learned directly from pixels (Athalye et al., 2026). Lastly, multiple approaches (Han et al., 2024; Liang et al., 2025) have used foundation models to learn symbolic representations of robot skills, but require extensive feedback or prior knowledge from human experts. TAMP has long been used to solve complex robot tasks (Dantam et al., 2018; Shah et al., 2020; Garrett et al., 2021). However, these approaches require symbolic models of robot skills for task planning. Various approaches have been developed to learn such symbolic models compatible with TAMP solvers from high-dimensional inputs (Konidaris et al., 2018; Silver et al., 2023; Shah et al., 2025). Additionally, the abstract representations learned through these methods are often not directly human-interpretable. SkillWrapper also connects to other domains in robotics, and the full related work can be found in Appendix J.

6 Conclusion

We characterize formal conditions under which abstract models support sound and complete task-level planning, and use them to derive principled criteria for generative predicate invention in SkillWrapper. Our method autonomously learns human-interpretable symbolic representations of black-box skills and produces operators directly usable by classical planners, without any predefined predicates, operators, or tasks for data collection. As foundation models improve in commonsense reasoning and visual grounding, SkillWrapper can naturally benefit from these advances, while its formally derived criteria continue to direct predicate invention toward the correct abstractions. Empirical results in simulation and on real robots show that SkillWrapper enables efficient long-horizon planning without hand-engineered abstractions, offering a promising path toward scalable symbolic model learning.

7 Reproducibility Statement

We provide full algorithm pseudocode in Appendix A, model definitions and proofs in Appendices B–D, operator-learning details and learned models in Appendices F–G, VLM reliability evaluations in Appendix H, and experimental setups, hyperparameters, and prompts in Appendix I. We additionally provide anonymized source code in the supplementary material. Robotouille experiments can be reproduced using the provided implementation, subject to availability and version stability of the external foundation-model API (i.e., OpenAI's GPT-5 (OpenAI, 2025)); exact reproduction of the physical-robot experiments additionally depends on the corresponding hardware.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, et al. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *Proceedings of the 6th Conference on Robot Learning*, 2022.

Michael Ahn, Debidatta Dwibedi, Chelsea Finn, Montserrat Gonzalez Arenas, Keerthana Gopalakrishnan, Karol Hausman, Brian Ichter, et al. AutoRT: Embodied Foundation Models for Large Scale Orchestration of Robotic Agents. In *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*, 2024.

Ankuj Arora, Humbert Fiorino, Damien Pellier, Marc Métivier, and Sylvie Pesty. A Review of Learning Planning Action Models. *The Knowledge Engineering Review*, 33:e20, 2018.

Ashay Athalye, Nishanth Kumar, Tom Silver, Yichao Liang, Jiuguang Wang, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. From Pixels to Predicates: Learning Symbolic World Models via Pretrained VLMs. *IEEE Robotics and Automation Letters (RA-L)*, 11(4):4002–4009, 2026.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Boyuan Chen, Zhuo Xu, Sean Kirmani, Brian Ichter, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. SpatialVLM: Endowing Vision-Language Models with Spatial Reasoning Capabilities. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

William Chen, Oier Mees, Aviral Kumar, and Sergey Levine. Vision-Language Models Provide Promptable Representations for Reinforcement Learning. *Transactions on Machine Learning Research (TMLR)*, 2025.

Sijie Cheng, Zhicheng Guo, Jingwen Wu, Kechen Fang, Peng Li, Huaping Liu, and Yang Liu. EgoThink: Evaluating First-Person Perspective Thinking Capability of Vision-Language Models. In *Proceedings of the 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal Manipulation Interface: In-The-Wild Robot Teaching Without In-The-Wild Robots. In *Proceedings of Robotics: Science and Systems*, 2024.

Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research*, 37(10):1134–1151, 2018.

S. Doncieux, D. Filliat, N. Díaz-Rodríguez, T. Hospedales, R. Duro, A. Coninx, D.M. Roijers, B. Girard, N. Perrin, and O. Sigaud. Open-ended learning: a conceptual framework based on representational redescription. *Frontiers in Neurorobotics*, 2018.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-E: An Embodied Multimodal Language Model. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Kuan Fang, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Adaptive Procedural Task Generation for Hard-Exploration Problems. In *Proceedings of the 9th International Conference on Learning Representations*, 2021.

Kuan Fang, Toki Migimatsu, Ajay Mandlekar, Li Fei-Fei, and Jeannette Bohg. Active Task Randomization: Learning Robust Skills via Unsupervised Generation of Diverse and Feasible Tasks. *Proceedings of the 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2022.

Kuan Fang, Fangchen Liu, Pieter Abbeel, and Sergey Levine. MOKA: Open-World Robotic Manipulation through Mark-Based Visual Prompting. *Proceedings of Robotics: Science and Systems*, 2024.

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.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016. doi: 10.1017/CBO9781139583923.

Gonzalo Gonzalez-Pumariega, Leong Su Yean, Neha Sunkara, and Sanjiban Choudhury. Robotouille: An Asynchronous Planning Benchmark for LLM Agents. In *Proceedings of the 13th International Conference on Learning Representations*, 2025.

Jiayuan Gu, Devendra Singh Chaplot, Hao Su, and Jitendra Malik. Multi-skill Mobile Manipulation for Object Rearrangement. In *Proceedings of the 11th International Conference on Learning Representations*, 2022.

Huy Ha, Pete Florence, and Shuran Song. Scaling Up and Distilling Down: Language-Guided Robot Skill Acquisition. In *Proceedings of the 7th Conference on Robot Learning*, 2023.

Muzhi Han, Yifeng Zhu, Song-Chun Zhu, Ying Nian Wu, and Yuke Zhu. InterPreT: Interactive Predicate Learning from Language Feedback for Generalizable Task Planning. In *Proceedings of Robotics: Science and Systems*, 2024.

Yining Hong, Haoyu Zhen, Peihao Chen, Shuhong Zheng, Yilun Du, Zhenfang Chen, and Chuang Gan. 3D-LLM: Injecting the 3D World into Large Language Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Yingdong Hu, Fanqi Lin, Tong Zhang, Li Yi, and Yang Gao. Look Before You Leap: Unveiling the Power of GPT-4V in Robotic Vision-Language Planning. *arXiv preprint arXiv:2311.17842*, 2023.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner Monologue: Embodied Reasoning through Planning with Language Models. In *Proceedings of the 6th Conference on Robot Learning*, 2023.

Steven James, Benjamin Rosman, and George Konidaris. Autonomous Learning of Object-Centric Abstractions for High-Level Planning. In *Proceedings of the 10th International Conference on Learning Representation*, 2022.

Hanxiao Jiang, Binghao Huang, Ruihai Wu, Zhuoran Li, Shubham Garg, Hooshang Nayyeri, Shenlong Wang, and Yunzhu Li. RoboEXP: Action-Conditioned Scene Graph via Interactive Exploration for Robotic Manipulation. In *Proceedings of the 8th Conference on Robot Learning*, 2025.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized Level Replay. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Brendan Juba, Hai S. Le, and Roni Stern. Safe Learning of Lifted Action Models. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 11 2021.

Michael Katz and Junkyu Lee. K* and partial order reduction for top-quality planning. In *Proceedings of the 16th Annual Symposium on Combinatorial Search*. AAAI Press, 2023.

George Konidaris. On the Necessity of Abstraction. *Current Opinion in Behavioral Sciences*, 29: 1–7, 2019. ISSN 2352-1546.

George Konidaris and Andrew Barto. Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining. In *Advances in Neural Information Processing Systems (NIPS)*, volume 22, 2009.

George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Pérez. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.

Leonardo Lamanna, Luciano Serafini, Mohamadreza Faridghasemnia, Alessandro Saffiotti, Alessandro Saetti, Alfonso Gerevini, and Paolo Traverso. Planning for Learning Object Properties. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

Amber Li and Tom Silver. Embodied Active Learning of Relational State Abstractions for Bilevel Planning. In *Proceedings of The 2nd Conference on Lifelong Learning Agents*, 2023.

Zhaoyi Li, Kelin Yu, Shuo Cheng, and Danfei Xu. LEAGUE++: Empowering Continual Robot Learning via Guided Skill Acquisition with Large Language Models. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024.

Yichao Liang, Nishanth Kumar, Hao Tang, Adrian Weller, Joshua B. Tenenbaum, Tom Silver, Joao F. Henriques, and Kevin Ellis. VisualPredicator: Learning Abstract World Models with Neuro-Symbolic Predicates for Robot Planning. In *Proceedings of the 13th International Conference on Learning Representations*, 2025.

Yichao Liang, Dat Nguyen, Cambridge Yang, Tianyang Li, Joshua B Tenenbaum, Carl Edward Rasmussen, Adrian Weller, Zenna Tavares, Tom Silver, and Kevin Ellis. ExoPredicator: Learning Abstract Models of Dynamic Worlds for Robot Planning. In *Proceedings of the 14th International Conference on Learning Representations*, volume 2026, pp. 49174–49214, 2026.

Arjun Majumdar, Anurag Ajay, Xiaohan Zhang, Pranav Putta, Sriram Yenamandra, Mikael Henaff, Sneha Silwal, Paul Mcvay, Oleksandr Maksymets, Sergio Arnaud, Karmesh Yadav, Qiyang Li, Ben Newman, Mohit Sharma, Vincent Berges, Shiqi Zhang, Pulkit Agrawal, Yonatan Bisk, Dhruv Batra, Mrinal Kalakrishnan, Franziska Meier, Chris Paxton, Alexander Sax, and Aravind Rajeswaran. OpenEQA: Embodied Question Answering in the Era of Foundation Models. In *Proceedings of the 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Toki Migimatsu and Jeannette Bohg. Grounding Predicates through Actions. In *Proceedings of the 2022 International Conference on Robotics and Automation*, 2022.

Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. EmbodiedGPT: Vision-Language Pre-Training via Embodied Chain of Thought. In *Advances in Neural Information Processing Systems*, 2024.

Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, Jacky Liang, Ishita Dasgupta, Annie Xie, Danny Driess, Ayzaan Wahid, Zhuo Xu, Quan Vuong, Tingnan Zhang, Tsang-Wei Edward Lee, Kuang-Huei Lee, Peng Xu, Sean Kirmani, Yuke Zhu, Andy Zeng, Karol Hausman, Nicolas Heess, Chelsea Finn, Sergey Levine, and Brian Ichter. PIVOT: Iterative Visual Prompting Elicits Actionable Knowledge for VLMs. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

OpenAI. Introducing GPT-5, 2025. URL `https://openai.com/index/introducing-gpt-5/`.

Shreyas Sundara Raman, Vanya Cohen, Ifrah Idrees, Eric Rosen, Ray Mooney, Stefanie Tellex, and David Paulius. CAPE: Corrective Actions from Precondition Errors using Large Language Models. In *Proceedings of the 2024 IEEE International Conference on Robotics and Automation*, 2024.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Robot Task Planning. In *Proceedings of the 7th Conference on Robot Learning*, 2023.

Allen Z. Ren, Jaden Clark, Anushri Dixit, Masha Itkina, Anirudha Majumdar, and Dorsa Sadigh. Explore until Confident: Efficient Exploration for Embodied Question Answering. In *Proceedings of Robotics: Science and Systems*, 2024.

Pierre Sermanet, Tianli Ding, Jeffrey Zhao, Fei Xia, Debidatta Dwibedi, Keerthana Gopalakrishnan, Christine Chan, Gabriel Dulac-Arnold, Sharath Maddineni, Nikhil J Joshi, et al. RoboVQA: Multimodal Long-Horizon Reasoning for Robotics. In *Proceedings of the 2024 IEEE International Conference on Robotics and Automation*. IEEE, 2024.

Naman Shah, Deepak Kala Vasudevan, Kislay Kumar, Pranav Kamojjhala, and Siddharth Srivastava. Anytime Integrated Task and Motion Policies for Stochastic Environments. In *Proceedings of the 2020 IEEE International Conference on Robotics and Automation*. IEEE, 2020.

Naman Shah, Jayesh Nagpal, and Siddharth Srivastava. From Real World to Logic and Back: Learning Generalizable Relational Concepts For Long Horizon Robot Planning. In *Proceedings of the 9th Conference on Robot Learning*, 2025.

Tom Silver, Rohan Chitnis, Joshua Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning Symbolic Operators for Task and Motion Planning. In *Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3182–3189, 2021.

Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Kaelbling, and Joshua B Tenenbaum. Predicate Invention for Bilevel Planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 12120–12129, 2023.

Theodore Sumers, Kenneth Marino, Arun Ahuja, Rob Fergus, and Ishita Dasgupta. Distilling Internet-Scale Vision-Language Models into Embodied Agents. In *Proceedings of the Fortieth International Conference on Machine Learning*, 2023.

Dídac Surís, Sachit Menon, and Carl Vondrick. ViperGPT: Visual Inference via Python Execution for Reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimír Vondruš, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training Home Assistants to Rearrange their Habitat. In *Advances in Neural Information Processing Systems*, 2021.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the Planning Abilities of Large Language Models - A Critical Investigation. In *Advances in Neural Information Processing Systems*, 2023.

Pulkit Verma, Shashank Rao Marpally, and Siddharth Srivastava. Discovering User-Interpretable Capabilities of Black-Box Planning Agents. In *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning*, 2022.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models. *Transactions on Machine Learning Research*, 2024a.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions. *arXiv preprint arXiv:1901.01753*, 2019.

Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. RoboGen: Towards Unleashing Infinite Data for Automated Robot Learning via Generative Simulation. In *Proceedings of the 41st International Conference on Machine Learning*, 2024b.

Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. FoundationPose: Unified 6D Pose Estimation and Tracking of Novel Objects. In *Proceedings of the 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Kai Xi, Stephen Gould, and Sylvie Thiébaux. Neuro-Symbolic Learning of Lifted Action Models from Visual Traces. *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 34(1), May 2024.

Jingkang Yang, Yuhao Dong, Shuai Liu, Bo Li, Ziyue Wang, Haoran Tan, Chencheng Jiang, Jiamu Kang, Yuanhan Zhang, Kaiyang Zhou, et al. Octopus: Embodied Vision-Language Programmer from Environmental Feedback. In *Proceedings of the 2024 European Conference on Computer Vision*, 2024.

Naoki Yokoyama, Alex Clegg, Joanne Truong, Eric Undersander, Tsung-Yen Yang, Sergio Arnaud, Sehoon Ha, Dhruv Batra, and Akshara Rai. ASC: Adaptive Skill Coordination for Robotic Mobile Manipulation. *IEEE Robotics and Automation Letters*, 9(1):779–786, 2024.

Bin Yu. Rates of convergence for empirical processes of stationary mixing sequences. *The Annals of Probability*, pp. 94–116, 1994.

Appendices

Contents

A Algorithms **17**

A.1 Active Data Collection 17

A.2 Predicate Invention 18

A.3 Operator Learning 19

B Properties of Learned Models **19**

C Proofs **21**

C.1 Establishing Soundness and Completeness 21

C.2 Properties of SkillWrapper 21

C.3 Illustration of Predicate Invention in Low-level Space 26

C.4 Operator Learning 27

D Additional Details on the Hypothesis Class and Sample Complexity **28**

D.1 Upper Bound on $|\mathcal{H}|$ 29

D.2 Sample Complexity for a Target (ϵ, δ) 29

E Comparisons of Symbolic Model Learning Methods **32**

F Operator Learning Details **33**

F.1 Lifting and Substitution 33

F.2 Running Example 33

G Case Studies, Learned Operators, and Example Tasks **36**

G.1 Result Analysis of Baseline Comparisons 36

G.2 Learned Predicates & Operators 37

G.3 Example Tasks 49

H VLM Reliability Study **51**

H.1 Classification Accuracy 51

H.2 Real-world Robustness 52

H.3 Other VLMs 55

I Implementation Details **58**

I.1 Planner and Planning Time 58

I.2 API Call 58

I.3 Hyperparameters 58

I.4 Experiments 58

I.5 Prompts 60

J Related Works **63**

K Glossary of Notation **64**

L AI Usage Disclosure **65**

A ALGORITHMS

Algorithm 1 SKILLWRAPPER

Input: Set of skills Ω, number of iterations $m \in \mathbb{N}_1$
Output: Abstract transition model $\mathcal{M} = \langle \mathcal{P}, \mathcal{A} \rangle$
$\mathcal{D}, \mathcal{P}, \mathcal{A} \leftarrow \emptyset$
for $i \in \{1, \ldots, m\}$ **do**
  $\underline{\boldsymbol{\omega}} \leftarrow$ PROPOSESKILLSEQ$(\Omega, \mathcal{D}, \mathcal{P}, \mathcal{A})$ ▷ Section 3.1
  $\mathcal{D} \leftarrow \mathcal{D} \bigcup$ EXECUTESKILLS$(\underline{\boldsymbol{\omega}})$
  $\mathcal{P} \leftarrow$ INVENTPREDICATES$(\Omega, \mathcal{D}, \mathcal{P}, \mathcal{A})$ ▷ Section 3.2
  $\mathcal{A} \leftarrow$ LEARNOPERATORS$(\mathcal{D}, \mathcal{P})$ ▷ Section 3.3
return $\mathcal{M} = \langle \mathcal{P}, \mathcal{A} \rangle$

A.1 ACTIVE DATA COLLECTION

Algorithm 2 Propose Skill Sequences

Input: Skill set Ω, dataset $\mathcal{D}$, predicate set $\mathcal{P}$, abstract actions $\mathcal{A}$, batch size n
Output: Proposed skill sequence $\underline{\boldsymbol{\omega}}$
seq_batch $\leftarrow$ GENERATESKILLSEQUENCES(Ω, n)
Scores $\leftarrow \{\}$
for $\underline{\boldsymbol{\omega}}$ **in** seq_batch **do**
  $b \leftarrow$ BALANCE$(\underline{\boldsymbol{\omega}})$
  $c \leftarrow$ COVERAGE$(\underline{\boldsymbol{\omega}})$
  Scores$[\underline{\boldsymbol{\omega}}] \leftarrow (b, c)$
return PARETOOPTIMAL(Scores)

Algorithm 3 Coverage

Input: Dataset $\mathcal{D}$, proposed skill sequence $\underline{\boldsymbol{\omega}}$
Output: Coverage score c
$\mathcal{Q} \leftarrow$ zero matrix of size $|\Omega| \times |\Omega|$ ▷ Construct a matrix of skill-pair counts
for $\langle s_i, \underline{\omega}_i, s_i' \rangle, \langle s_{i+1}, \underline{\omega}_{i+1}, s_{i+1}' \rangle$ in $\mathcal{D}$ **do** ▷ Iterate over all consecutive pairs of transitions
  $\mathcal{Q}[\underline{\omega}_i, \underline{\omega}_{i+1}] \leftarrow \mathcal{Q}[\underline{\omega}_i, \underline{\omega}_{i+1}] + 1$
$\mathcal{Q}' \leftarrow \mathcal{Q}$ ▷ New skill-pair count initialized
for $\langle \underline{\omega}_i, \underline{\omega}_{i+1} \rangle$ in $\underline{\boldsymbol{\omega}}$ **do**
  $\mathcal{Q}'[\underline{\omega}_i, \underline{\omega}_{i+1}] \leftarrow \mathcal{Q}'[\underline{\omega}_i, \underline{\omega}_{i+1}] + 1$
$c \leftarrow$ ENTROPY$(\mathcal{Q}')$ − ENTROPY$(\mathcal{Q})$
return c

Algorithm 4 Balance

Input: Operator set $\mathcal{A}$, proposed skill sequence $\underline{\boldsymbol{\omega}}$, target ratio r^*
Output: Balance score b
exec_count $\leftarrow 0$ ▷ Total number of executable skills
sequence_length $\leftarrow$ LENGTH($\underline{\boldsymbol{\omega}}$)
seq $\leftarrow \{\alpha(s_0)\}$ ▷ Store the trace of after-execution state
for $\underline{\omega}_i$ in $\underline{\boldsymbol{\omega}}$ **do**
  $\underline{\boldsymbol{a}} \leftarrow \{\}$
  for $\underline{a} \in \underline{\mathcal{A}}_{\underline{\omega}_i}$ **do**
    if $\alpha(\text{seq}[-1]) \models \underline{\text{PRE}}_{\underline{a}}$ **then** ▷ Successful execution predicted by the current model
      $\underline{\boldsymbol{a}} \leftarrow \underline{\boldsymbol{a}} \bigcup \{\underline{a}\}$
  $\hat{\underline{a}} = \arg\max_{\underline{a} \in \underline{\boldsymbol{a}}} |\text{PRE}_{\underline{a}}|$
  $s^{\uparrow}_{new} \leftarrow$ APPLYOPERATOR($\hat{\underline{a}}$) ▷ Calculate the abstract state after execution
  seq $\leftarrow$ seq $\bigcup \{s^{\uparrow}_{new}\}$ ▷ Append current low-level state to the trace
  exec_count $\leftarrow$ exec_count $+ 1$
$\hat{r} \leftarrow$ exec_count/sequence_length
$b \leftarrow -|\hat{r} - r^*|$
return b

A.2 PREDICATE INVENTION

Algorithm 5 Invent Predicates

Input: Skill set Ω, transition dataset $\mathcal{D}(\omega) = \{\langle s, \underline{\omega}, s' \rangle\}_{\omega}$, existing predicate set $\mathcal{P}$, abstract actions $\mathcal{A}$.
Output: Predicate set $\mathcal{P}$
for $\omega \in \Omega$ **do**
  while precondition inconsistency exists **do** ▷ equation 1
    $p \leftarrow$ NEWPREDICATE
    $\mathcal{P} \leftarrow \mathcal{P} \bigcup \{p\}$ **if** SCOREPRECOND($p, \mathcal{P}, \omega, \mathcal{D}$)
  while effect inconsistency exists **do** ▷ equation 2
    $p \leftarrow$ NEWPREDICATE
    $\mathcal{P} \leftarrow \mathcal{P} \bigcup \{p\}$ **if** SCOREEFF($p, \mathcal{P}, \omega, \mathcal{D}$)
return $\mathcal{P}$

Algorithm 6 Scoring Functions for Invented Predicates

Input: Candidate predicate $\hat{p}$, existing predicate set $\mathcal{P}$, skill ω, and transition dataset $\mathcal{D}$.
Parameters: Threshold h

ScorePrecond:
$\hat{\mathcal{P}} \leftarrow \mathcal{P} \bigcup \{\hat{p}\}$
$\hat{\mathcal{A}} \leftarrow$ LearnOperators$(\mathcal{D}, \hat{\mathcal{P}})$ ▷ Hypothetical operators after including $\hat{p}$
total $\leftarrow 0$
valid $\leftarrow 0$
for $\langle s, \underline{\omega}, s' \rangle \in \mathcal{D}$ **do**
  if $\exists\, \underline{\hat{a}} \models \underline{\hat{\mathcal{A}}_{\omega}}$, s.t. $\alpha(s) \in \underline{\text{Pre}_{\hat{a}}}$, and $s \in \mathcal{I}_{\underline{\omega}}$ **then**
    valid $\leftarrow$ valid $+ 1$
  if $\nexists\, \underline{\hat{a}} \models \underline{\hat{\mathcal{A}}_{\omega}}$, s.t. $\alpha(s) \in \underline{\text{Pre}_{\hat{a}}}$, and $s \notin \mathcal{I}_{\underline{\omega}}$ **then**
    valid $\leftarrow$ valid $+ 1$
  total $\leftarrow$ total $+ 1$
return valid/total $> h$

ScoreEff:
$\hat{\mathcal{P}} \leftarrow \mathcal{P} \bigcup \{p\}$
$\hat{\mathcal{A}} \leftarrow$ LearnOperators$(\mathcal{D}, \hat{\mathcal{P}})$
total $\leftarrow 0$
valid $\leftarrow 0$
for $\langle s, \underline{\omega}, s' \rangle \in \mathcal{D}$ **do**
  if $\exists\, \hat{a} \in \hat{\mathcal{A}}$, s.t. $\alpha_{\hat{\mathcal{P}}}(\mathrm{s}') \setminus \alpha_{\hat{\mathcal{P}}}(\mathrm{s}) = \underline{\text{Eff}_{\hat{a}}}$, and s $\in \beta_{\underline{\omega}}$ **then**
    valid $\leftarrow$ valid $+ 1$
  if $\nexists\, \hat{a} \in \hat{\mathcal{A}}$, s.t. $\alpha_{\hat{\mathcal{P}}}(\mathrm{s}') \setminus \alpha_{\hat{\mathcal{P}}}(\mathrm{s}) = \underline{\text{Eff}_{\hat{a}}}$, and s $\notin \beta_{\underline{\omega}}$ **then**
    valid $\leftarrow$ valid $+ 1$
  total $\leftarrow$ total $+ 1$
return valid/total $> h$

A.3 Operator Learning

Algorithm 7 Learn Operators

Input: Dataset $\mathcal{D}(\omega) = \{\langle s, \underline{\omega}, s' \rangle\}_{\omega}$, predicate set $\mathcal{P}$
Output: Abstract actions $\mathcal{A}$
eff_dict $\leftarrow$ defaultdict() ▷ Store clustered effects
for each successful transition $\tau = \langle s, \underline{\omega}, s' \rangle \in \mathcal{D}$ **do**
  Eff $\leftarrow \Lambda(\Delta(s, s'))$
  eff_dict[Eff] $\leftarrow$ eff_dict[Eff] $\bigcup \{\langle s, \underline{\omega}, s' \rangle\}$
$\mathcal{A} \leftarrow \emptyset$
for Eff $\in$ eff_dict **do**
  Pre $\leftarrow \Pi_{\langle s, \underline{\omega}, s' \rangle \in \text{eff_dict[Eff]}} \alpha(s)$
  $a \leftarrow \langle \text{Pre}, \text{Eff} \rangle$
  $\mathcal{A} \leftarrow \mathcal{A} \bigcup \{a\}$
return $\mathcal{A}$

B Properties of Learned Models

Relational predicates are the basic units of the abstract representation of the low-level state space. In this section, we characterize the conditions of the learned representations of a finite set of skills using relational predicates to support high-level planning, in the context of model learning. From here, chaining the skills is enabled by applying predicates from the representation of each low-level skill to others for clustering (described in Section 3.3).

Effective skill planning requires an accurate abstract model and grounding functions. All forms of abstractions are typically lossy, i.e., while learning an abstract transition model, certain low-level

environment details may not be captured. Conversely, the learned model must accurately retain the information needed to produce plans to guarantee soundness and completeness. In this section, we characterize the conditions under which an abstract model facilitates correct, sound, and complete planning, for the purpose of constraining how we can *construct* such a model.

To begin formalizing the relationship between a skill $\omega \in \Omega$ and some abstract model $\mathcal{M}$, we define two sets, Φ_ω and Ξ_ω, representing the states in which the model predicts that the successful skill execution may either be initiated (when $s \in \Phi_\omega$) or terminated (when $s \in \Xi_\omega$), respectively.

Definition 1. *Given a skill instance $\underline{\omega} \in \underline{\Omega}$ and abstract model $\mathcal{M}$, we define $\Phi_{\underline{\omega}}$ and $\Xi_{\underline{\omega}}$ as follows:*

$$\Phi_{\underline{\omega}} \coloneqq \bigcup_{\underline{a} \in \underline{\mathcal{A}_\omega}} \Gamma(\underline{\textit{PRE}_a}) \tag{3}$$

$$\Xi_{\underline{\omega}} \coloneqq \bigcup_{\underline{a} \in \underline{\mathcal{A}_\omega}} \Gamma\big((\underline{\textit{PRE}_a} \setminus \underline{\textit{EFF}_a^-}) \cup \underline{\textit{EFF}_a^+}\big) \tag{4}$$

For a non-instantiated skill $\omega \in \Omega$, we define $\Phi_\omega = \cup_{\underline{\omega}} \Phi_{\underline{\omega}}$ and $\Xi_\omega = \cup_{\underline{\omega}} \Xi_{\underline{\omega}}$ over all possible skill instances with type-compatible objects.

We define a *correct* abstract model as one that perfectly captures the pre-success set and success set of all skills.

Definition 2 (Correct Model)**.** *Let $\mathcal{M} = (\mathcal{P}, \mathcal{A})$ be a model for environment $\langle \mathcal{S}, \mathsf{T}, \Omega, T \rangle$ and the set of skills Ω, and let Φ_ω and Ξ_ω be the approximate pre-success and success sets (Def. 1). The model $\mathcal{M}$ is a **correct model** iff:*

$$\forall \omega \in \Omega, s \in \mathcal{S}\colon \big(s \in \mathcal{I}_\omega \iff s \in \Phi_\omega\big) \wedge \big(s \in \beta_\omega \iff s \in \Xi_\omega\big). \tag{5}$$

A correct model exactly captures the pre-success and success sets of all skills, making correctness an idealized target for representation learning. We next define the plan-level properties of *soundness* and *completeness* and relate them to correctness.

A symbolic model is *sound* if it correctly predicts the effects of a plan: whenever a complete and sound planner predicts that a sequence of skills will reach some abstract state, executing the corresponding skills in the environment truly leads there. Soundness rules out spurious symbolic transitions that do not correspond to realizable outcomes. Formally, we introduce the settings as follows: Let $\mathcal{M} = (\mathcal{P}, \mathcal{A})$ denote an abstract transition model, where $\mathcal{P}$ is a finite set of predicates defining an *abstract state space* $\mathcal{S}^\uparrow$, and $\mathcal{A}$ is a set of *abstract actions* with preconditions and effects expressed in terms of $\mathcal{P}$. Each abstract state $s^\uparrow \in \mathcal{S}^\uparrow$ is obtained by an *abstraction function* $\alpha : \mathcal{S} \to \mathcal{S}^\uparrow$ that maps low-level states $s \in \mathcal{S}$ to truth assignments over $\mathcal{P}$. Given a skill planning problem $\mathbf{p}$, a symbolic planner can produce an abstract plan $\underline{\boldsymbol{a}}$ using the learned model $\mathcal{M}$, and the skill plan $\underline{\boldsymbol{\omega}}$ is then obtained by mapping all operators in $\underline{\boldsymbol{a}}$ back to their corresponding skills.

Definition 3 (Soundness)**.** *For an abstract plan $\mathbf{a} = [a_1, \ldots, a_k]$ with corresponding skill plan $\boldsymbol{\omega} = [\omega_1, \ldots, \omega_k]$ for a planning problem $\mathbf{p} = \langle \mathcal{O}, s_0, \mathcal{S}_g \rangle$, let $s_i = T(s_{i-1}, \omega_i)$. The model $\mathcal{M}$ is* sound *iff, for every i,*

$$s_{i-1} \in I_{\omega_i} \quad \textit{and} \quad \alpha(s_i) = \bar{T}_M(\alpha(s_{i-1}), a_i),$$

where $T_M(\alpha(s_{i-1}), a_i)$ is the abstract state predicted by $\mathcal{M}$ after executing a_i from $\alpha(s_{i-1})$. We define soundness over abstract plans whose execution does not encounter an effect-ambiguous state, in which multiple operators of the same skill are simultaneously applicable but predict distinct effects: $\exists a_i, a_j \in \mathcal{A}_\omega : \quad s^\uparrow \models \textit{PRE}_{a_i},\ s^\uparrow \models \textit{PRE}_{a_j},\ \textit{EFF}_{a_i} \neq \textit{EFF}_{a_j}$.

A symbolic model is *complete* if it never omits real solutions: whenever the environment admits a way to solve a skill planning problem, the planner can find a corresponding abstract plan in the model. Completeness rules out gaps in symbolic coverage that would make feasible problems appear unsolvable.

Definition 4 (Completeness)**.** *The model $\mathcal{M}$ is* complete *iff, for any skill planning problem $\mathbf{p} = \langle \mathcal{O}, s_0, \mathcal{S}_g \rangle$ and any skill plan $\underline{\boldsymbol{\omega}}$ that achieves the goal states $\mathcal{S}_g$ from an initial state s_0, there exists an abstract plan $\underline{\boldsymbol{a}}$ over $\mathcal{M}$ such that*

$$\bar{T}_{\mathcal{M}}(\underline{\boldsymbol{a}}, \alpha(s_0)) \models \alpha(\mathcal{S}_g), \quad \alpha(S_g) \coloneqq \{\alpha(s) : s \in S_g\}.$$

Although methods for constructing such models exist and have been investigated in previous work in the bottom-up setting, no existing work in the generative predicate invention setting provides or even discusses these properties in their methods. We next establish the relationship between these model properties and analyze how SkillWrapper approaches them through its predicate-invention and operator-learning procedures.

C Proofs

C.1 Establishing Soundness and Completeness

We now relate the correctness property defined in Appendix B to soundness and completeness. SkillWrapper can represent distinct successful conditional effects of the same skill using separate operators. When the preconditions of such operators overlap, multiple operators may be simultaneously applicable while predicting different effects, in which case the abstract model does not uniquely specify the realized transition. As defined in Appendix B, we therefore consider soundness over effect-unambiguous plans. Under this definition, correctness is sufficient to establish both soundness and completeness.

Proposition 1 (Correctness implies soundness and completeness)**.** *Any correct abstract model $\mathcal{M}$ learned by* SkillWrapper *is sound and complete.*

Proof. Since $\mathcal{M}$ is correct, for every skill ω its estimated pre-success and success sets exactly match the true sets:

$$\Phi_\omega = I_\omega, \qquad \Xi_\omega = \beta_\omega.$$

For soundness, consider any valid abstract plan $\mathbf{a} = [a_1, \ldots, a_n]$ and its corresponding skill sequence $\boldsymbol{\omega} = [\omega_1, \ldots, \omega_n]$, starting from s_0. Since a_i is applicable at $\alpha(s_{i-1})$, $s_{i-1} \in \Gamma(\textsc{Pre}_{a_i}) \subseteq \Phi_{\omega_i}$. By correctness, $s_{i-1} \in I_{\omega_i}$, so each corresponding skill execution succeeds. By construction, SkillWrapper first clusters successful transitions by their abstract effects and then computes each operator's preconditions from the initial states of the same cluster. Thus, the precondition and effect of each operator remain associated through their supporting transitions; under effect-unambiguity, an applicable operator predicts the corresponding abstract effect. Thus, for every step,

$$\alpha(s_i) = \bar{T}_{\mathcal{M}}(\alpha(s_{i-1}), a_i), \qquad s_i = T(s_{i-1}, \omega_i).$$

Applying this correspondence inductively over the sequence shows that the abstract plan correctly predicts its corresponding skill execution, establishing soundness.

For completeness, consider any feasible skill plan $\boldsymbol{\omega} = [\omega_1, \ldots, \omega_n]$ that reaches a goal state from s_0. At every step, feasibility gives $s_{i-1} \in I_{\omega_i} = \Phi_{\omega_i}$, so by the definition of Φ_{ω_i} there exists an operator $a_i \in \mathcal{A}_{\omega_i}$ whose preconditions are satisfied. Since the execution is successful and $\beta_{\omega_i} = \Xi_{\omega_i}$, its successful abstract effect is represented by an operator learned from the corresponding effect cluster, whose preconditions are derived from that same cluster. Choosing this corresponding operator at each step constructs an abstract plan $[a_1, \ldots, a_n]$ for the feasible skill plan. Its final abstract state is $\alpha(s_n) \in \alpha(S_g)$, and hence the model is complete. □

Having established how correctness relates to soundness and completeness, we next analyze how SkillWrapper approaches correctness through its predicate-invention conditions and learned operators.

C.2 Properties of SkillWrapper

Lemma 1. *Let $\mathcal{M}$ be the model learned by* SkillWrapper *from dataset $\mathcal{D}$ after all contrastive transition pairs satisfying the predicate-invention conditions have been resolved. Then, for every observed transition $\langle s, \underline{\omega}, s' \rangle \in \mathcal{D}$, the estimated success set agrees with the true success set at s':*

$$s' \in \Xi_{\underline{\omega}} \iff s' \in \beta_{\underline{\omega}}.$$

Proof. Suppose, for contradiction, that the equivalence does not hold for some observed transition $\tau = \langle s, \underline{\omega}, s' \rangle \in \mathcal{D}$. Then exactly one of the following two cases must hold.

First, suppose $s' \in \Xi_{\underline{\omega}} \setminus \beta_{\underline{\omega}}$. By the definition of $\Xi_{\underline{\omega}}$, there exists an operator $\underline{a} \in \underline{\mathcal{A}}_{\underline{\omega}}$ such that

$$s' \in \Gamma\big((\textsc{Pre}_{\underline{a}} \setminus \underline{\textsc{Eff}}^-_{\underline{a}}) \cup \underline{\textsc{Eff}}^+_{\underline{a}}\big).$$

Since a is learned from a cluster of successful transitions sharing the same lifted abstract effect, while $s' \notin \beta_{\underline{\omega}}$, τ is a failed transition whose abstract change is not distinguished from that successful effect. This yields a contrastive pair satisfying Condition 2, contradicting the assumption that all such pairs have been resolved.

Conversely, suppose $s' \in \beta_{\underline{\omega}} \setminus \Xi_{\underline{\omega}}$. Then τ is a successful transition. Operator learning places τ in a successful effect cluster and constructs a corresponding operator $a \in \mathcal{A}_{\underline{\omega}}$. By construction, the STRIPS update of a contains s', i.e., $s' \in \Gamma\big((\textsc{Pre}_{\underline{a}} \setminus \underline{\textsc{Eff}}^-_{\underline{a}}) \cup \underline{\textsc{Eff}}^+_{\underline{a}}\big)$. Therefore $s' \in \Xi_{\underline{\omega}}$, again a contradiction.

Hence, for every $\langle s, \underline{\omega}, s' \rangle \in \mathcal{D}$,

$$s' \in \Xi_{\underline{\omega}} \iff s' \in \beta_{\underline{\omega}}.$$

□

Theorem 1 (**Empirical Consistency of** SkillWrapper**).** *Let $\mathcal{M}$ be the model learned by* SkillWrapper *from dataset $\mathcal{D}$ after all contrastive transition pairs satisfying the predicate-invention conditions have been resolved. Then, for every observed transition $\langle s, \underline{\omega}, s' \rangle \in \mathcal{D}$, the estimated pre-success and success sets agree with their true counterparts:*

$$s \in \Phi_{\underline{\omega}} \iff s \in \mathcal{I}_{\underline{\omega}}, \qquad s' \in \Xi_{\underline{\omega}} \iff s' \in \beta_{\underline{\omega}}.$$

Proof. We first show that the estimated pre-success set agrees with the true pre-success set on every observed initial state. Suppose otherwise for some transition $\tau_j = \langle s_j, \underline{\omega}_j, s'_j \rangle \in \mathcal{D}$. There are two cases.

(i) False positive. Suppose $s_j \notin \mathcal{I}_{\underline{\omega}_j}$ but $s_j \in \Phi_{\underline{\omega}_j}$. By the definition of Φ_ω, some operator associated with $\underline{\omega}_j$ is applicable at s_j. This operator is learned from a cluster containing at least one successful transition $\tau_i = \langle s_i, \underline{\omega}_i, s'_i \rangle$ of the same skill, for which $s_i \in \mathcal{I}_{\underline{\omega}_i}$ and $s_i \in \Phi_{\underline{\omega}_i}$. Thus, τ_i and τ_j satisfy Condition 1, contradicting the assumption that all such contrastive pairs have been resolved.

(ii) False negative. Suppose $s_j \in \mathcal{I}_{\underline{\omega}_j}$ but $s_j \notin \Phi_{\underline{\omega}_j}$. Then τ_j is a successful transition. Under a fixed dataset $\mathcal{D}$, operator learning places every successful transition into an effect cluster and constructs an operator whose preconditions are obtained by intersecting the abstract initial states in that cluster. Hence the initial state of every transition in the cluster satisfies the learned preconditions, implying $s_j \in \Phi_{\underline{\omega}_j}$, a contradiction. Further details of this operator-learning property are given in Appendix C.4.

Appendix C.3 provides the complete case-by-case analysis of the possible relations between Φ_ω and $\mathcal{I}_\omega$ and shows that the above two cases exhaust all observed pre-success inconsistencies.

Therefore, for every observed transition, $s \in \Phi_{\underline{\omega}}$ iff $s \in \mathcal{I}_{\underline{\omega}}$. By Lemma 1, we also have $s' \in \Xi_{\underline{\omega}}$ iff $s' \in \beta_{\underline{\omega}}$ on every observed terminal state. Together, these establish the claimed empirical consistency on $\mathcal{D}$. □

To account for temporal dependence in actively collected data, we model the transitions within each episode as a possibly nonstationary β-mixing process, with dependence between observations separated by k steps bounded by $\beta(k)$ (Yu, 1994). Within an episode, the active data-collection strategy is fixed.

For a fixed candidate model $\mathcal{M}$ and transition $\tau = \langle s, \underline{\omega}, s' \rangle$, define the error indicator and its episode-averaged expectation as

$$\ell_{\mathcal{M}}(\tau) \coloneqq \mathbf{1}\big\{ s \in \mathcal{I}_{\underline{\omega}} \ominus \Phi_{\underline{\omega}} \ \vee \ s' \in \beta_{\underline{\omega}} \ominus \Xi_{\underline{\omega}} \big\}, \qquad d_{\mathrm{compl}}(\mathcal{M}, \mathcal{M}^*) \coloneqq \frac{1}{n} \sum_{t=1}^{n} \mathbb{E}[\ell_{\mathcal{M}}(\tau_t)],$$

where $\ominus$ denotes symmetric difference. By Theorem 1, the learned model has zero empirical error on every observed transition. Since the active data-collection strategy is fixed within each episode and updated only between episodes, we first establish a single-episode bound in Lemma 2, and then extend the result across multiple episodes in Theorem 2.

Lemma 2. *Let $\mathcal{D}_n = \{\tau_t\}_{t=1}^n$ be collected within one episode under the fixed within-episode β-mixing process described above, and let $\mathcal{H}$ be a fixed finite model hypothesis class. For any $\epsilon \in (0,1)$ and integer $1 \le k \le n$, define $m = \lfloor n/k \rfloor$. With probability at least*

$$1 - |\mathcal{H}| \left(e^{-m\epsilon} + (m-1)\beta(k)\right),$$

every model $\mathcal{M} \in \mathcal{H}$ satisfying the empirical consistency conditions of Theorem 1 on $\mathcal{D}_n$ satisfies

$$d_{\mathrm{compl}}(\mathcal{M}, \mathcal{M}^*) \le \epsilon.$$

Proof. For any fixed $\mathcal{M} \in \mathcal{H}$, define

$$\mathrm{Err}(\mathcal{M}) \coloneqq d_{\mathrm{compl}}(\mathcal{M}, \mathcal{M}^*),$$

and let X_i be the binary indicator of whether an error occurs on the i-th transition:

$$X_i \coloneqq \ell_{\mathcal{M}}(\tau_i) = \mathbf{1}\left[(s_i \in \mathcal{I}_{\underline{\omega}_i} \ominus \Phi_{\underline{\omega}_i}) \vee (s_i' \in \beta_{\underline{\omega}_i} \ominus \Xi_{\underline{\omega}_i})\right].$$

The empirical error over the n observed transitions is

$$\widehat{\mathrm{Err}}(\mathcal{M}) \coloneqq \frac{1}{n}\sum_{i=1}^n X_i.$$

By definition,

$$\mathrm{Err}(\mathcal{M}) = \frac{1}{n}\sum_{i=1}^n \mathbb{E}[X_i].$$

Unlike in the stationary case, the expectations $\mathbb{E}[X_i]$ need not be identical across transition indices.

Suppose that some bad hypothesis $\mathcal{M} \in \mathcal{H}$ satisfies $\mathrm{Err}(\mathcal{M}) > \epsilon$. If $\widehat{\mathrm{Err}}(\mathcal{M}) = 0$, then $X_i = 0$ for every $i = 1, \dots, n$.

To handle temporal dependencies among consecutive transitions, fix a time lag $k \ge 1$ and let $m = \lfloor n/k \rfloor$. Write $p_i = \mathbb{E}[X_i]$ and partition the transition indices into the k interleaved subsequences

$$J_r = \{r, r+k, r+2k, \dots\} \cap \{1, \dots, n\}, \qquad r = 1, \dots, k.$$

Since these subsequences partition all n indices, there exists some J_{r^*} whose average error probability is at least the overall average:

$$\frac{1}{|J_{r^*}|}\sum_{i \in J_{r^*}} p_i \ge \frac{1}{n}\sum_{i=1}^n p_i = \mathrm{Err}(\mathcal{M}).$$

Since $|J_{r^*}| \ge m$, retain m indices $t_1 < \cdots < t_m$ in J_{r^*} with the largest values of p_i. Then $t_{j+1} - t_j \ge k$ and

$$\sum_{j=1}^m p_{t_j} \ge m\,\mathrm{Err}(\mathcal{M}) > m\epsilon.$$

Let $Y_j = X_{t_j}$ for $j = 1, \dots, m$. Because zero empirical error requires every observed transition to be correct,

$$\Pr\left(\widehat{\mathrm{Err}}(\mathcal{M}) = 0\right) \le \Pr(Y_1 = 0, \dots, Y_m = 0).$$

Applying the β-mixing coupling inequality to this k-spaced subsequence, we compare it with independent proxy variables $\widetilde{Y}_1, \dots, \widetilde{Y}_m$ having the same respective marginal distributions:

$$\Pr(Y_1 = 0, \dots, Y_m = 0) \le \Pr(\widetilde{Y}_1 = 0, \dots, \widetilde{Y}_m = 0) + (m-1)\beta(k).$$

The proxy variables need not be identically distributed. Since $\Pr(\widetilde{Y}_j = 1) = p_{t_j}$, independence gives

$$\begin{aligned}\Pr(\widetilde{Y}_1 = 0, \dots, \widetilde{Y}_m = 0) &= \prod_{j=1}^m (1 - p_{t_j}) \\ &\le \exp\left(-\sum_{j=1}^m p_{t_j}\right) \\ &\le e^{-m\mathrm{Err}(\mathcal{M})} \le e^{-m\epsilon}.\end{aligned}$$

Therefore, for any fixed $\mathcal{M} \in \mathcal{H}$ with $\mathrm{Err}(\mathcal{M}) > \epsilon$,

$$\Pr\Big(\widehat{\mathrm{Err}}(\mathcal{M}) = 0\Big) \le e^{-m\epsilon} + (m-1)\beta(k).$$

Applying a union bound over $\mathcal{H}$, we obtain

$$\Pr\Big[\exists\, \mathcal{M} \in \mathcal{H} : \mathrm{Err}(\mathcal{M}) > \epsilon \ \wedge\ \widehat{\mathrm{Err}}(\mathcal{M}) = 0\Big] \le |\mathcal{H}| \left(e^{-m\epsilon} + (m-1)\beta(k)\right).$$

Taking the complement, with probability at least

$$1 - |\mathcal{H}| \left(e^{-m\epsilon} + (m-1)\beta(k)\right),$$

every empirically consistent model $\mathcal{M} \in \mathcal{H}$ satisfies

$$d_{\mathrm{compl}}(\mathcal{M}, \mathcal{M}^*) \le \epsilon.$$

□

We now extend the single-episode result to multiple episodes, allowing the active data-collection strategy to change between episodes. Let $\tau_{r,t}$ denote transition t in episode r, and let $\mathcal{D}_{1:r-1}$ denote the preceding collection history. Conditional on this history, the collection strategy within episode r is fixed. Define the average conditional error across K episodes as

$$d_{\mathrm{compl},K}(\mathcal{M}, \mathcal{M}^*) := \frac{1}{K} \sum_{r=1}^{K} \mathbb{E}\left[\frac{1}{n} \sum_{t=1}^{n} \ell_{\mathcal{M}}(\tau_{r,t}) \,\middle|\, \mathcal{D}_{1:r-1}\right].$$

Lemma 2 gives a generalization bound within a single episode. Since the data-collection policy is updated only between episodes, we condition on the history before each episode and apply the lemma episode-wise, accumulating the error and mixing terms across episodes to obtain the following result.

Theorem 2 (**Asymptotic Convergence of** SkillWrapper)**.** *Let $\mathcal{H}$ be a fixed finite model hypothesis class, and let $\widehat{\mathcal{M}}_K \in \mathcal{H}$ satisfy the empirical consistency conditions of Theorem 1 on all transitions from K episodes of length n. For any $\epsilon \in (0,1)$ and integer $1 \le k \le n$, let $m = \lfloor n/k \rfloor$. With probability at least*

$$1 - |\mathcal{H}| \left(e^{-Km\epsilon} + K(m-1)\beta(k)\right),$$

the learned model $\widehat{\mathcal{M}}_K$ satisfies

$$d_{\mathrm{compl},K}(\widehat{\mathcal{M}}_K, \mathcal{M}^*) \le \epsilon.$$

In particular, choosing $k = n$ gives $m = 1$ and eliminates the mixing term, so $d_{\mathrm{compl},K}(\widehat{\mathcal{M}}_K, \mathcal{M}^*)$ converges to zero in probability as $K \to \infty$.

Proof. For a fixed model $\mathcal{M} \in \mathcal{H}$, define its conditional error in episode r as

$$R_r(\mathcal{M}) := \mathbb{E}\left[\frac{1}{n} \sum_{t=1}^{n} \ell_{\mathcal{M}}(\tau_{r,t}) \,\middle|\, \mathcal{D}_{1:r-1}\right].$$

By the definition of $d_{\mathrm{compl},K}$,

$$d_{\mathrm{compl},K}(\mathcal{M}, \mathcal{M}^*) = \frac{1}{K} \sum_{r=1}^{K} R_r(\mathcal{M}).$$

Fix $\mathcal{M} \in \mathcal{H}$ such that $d_{\mathrm{compl},K}(\mathcal{M}, \mathcal{M}^*) > \epsilon$. Then

$$\sum_{r=1}^{K} R_r(\mathcal{M}) > K\epsilon.$$

We apply Lemma 2 conditionally within each episode. Conditioning on the history $\mathcal{D}_{1:r-1}$ fixes the data-collection policy used in episode r, while the transitions within that episode remain a possibly nonstationary β-mixing process. Let $m = \lfloor n/k \rfloor$ and

$$q := (m-1)\beta(k).$$

Following the construction in Lemma 2, for episode r we obtain m k-spaced samples and independent proxy variables with the same respective marginals. Let B_r denote the event that the coupling between the dependent samples and their independent proxies fails. Then

$$\Pr(B_r \mid \mathcal{D}_{1:r-1}) \leq q.$$

Therefore, by the union bound,

$$\Pr\left(\bigcup_{r=1}^{K} B_r\right) \leq Kq = K(m-1)\beta(k).$$

Now consider the complementary event in which none of these couplings fails. Let C_r denote the event that all m independent proxy samples selected from episode r incur zero error under $\mathcal{M}$. The argument of Lemma 2 gives

$$\Pr(C_r \mid \mathcal{D}_{1:r-1}) \leq e^{-mR_r(\mathcal{M})}.$$

This bound remains valid although the policy changes between episodes, because it is conditioned on the complete history preceding episode r.

We next combine these bounds across episodes. Let the conditioning history include all data and proxy variables generated before the current episode. Since $R_r(\mathcal{M})$ is determined by the history before episode r, repeated conditional expectation gives

$$\mathbb{E}\left[\mathbf{1}[C_1 \cap \cdots \cap C_K] \exp\left(m \sum_{r=1}^{K} R_r(\mathcal{M})\right)\right] \leq 1.$$

Indeed, for each episode r,

$$e^{mR_r(\mathcal{M})} \Pr(C_r \mid \mathcal{D}_{1:r-1}) \leq e^{mR_r(\mathcal{M})} e^{-mR_r(\mathcal{M})} = 1,$$

so applying this inequality successively from episode K back to episode 1 yields the result above.

If $C_1, \ldots, C_K$ all occur and $d_{\mathrm{compl},K}(\mathcal{M}, \mathcal{M}^*) > \epsilon$, then

$$m \sum_{r=1}^{K} R_r(\mathcal{M}) > Km\epsilon.$$

Hence,

$$\Pr(C_1 \cap \cdots \cap C_K \wedge d_{\mathrm{compl},K}(\mathcal{M}, \mathcal{M}^*) > \epsilon) \leq e^{-Km\epsilon}.$$

If $\mathcal{M}$ has zero empirical error on all observed transitions, then it has zero error on every selected transition in every episode. Therefore, whenever none of the coupling events B_r occurs, zero empirical error implies $C_1 \cap \cdots \cap C_K$. Combining the two failure events above gives, for any fixed $\mathcal{M} \in \mathcal{H}$,

$$\Pr\left[d_{\mathrm{compl},K}(\mathcal{M}, \mathcal{M}^*) > \epsilon \wedge \widehat{\mathrm{Err}}(\mathcal{M}) = 0\right] \leq e^{-Km\epsilon} + K(m-1)\beta(k),$$

where $\widehat{\mathrm{Err}}(\mathcal{M})$ denotes the empirical error over all observed transitions.

Applying a union bound over the finite hypothesis class $\mathcal{H}$,

$$\Pr\left[\exists\, \mathcal{M} \in \mathcal{H} : d_{\mathrm{compl},K}(\mathcal{M}, \mathcal{M}^*) > \epsilon \wedge \widehat{\mathrm{Err}}(\mathcal{M}) = 0\right] \leq |\mathcal{H}| \left(e^{-Km\epsilon} + K(m-1)\beta(k)\right).$$

By Theorem 1, $\widehat{\mathcal{M}}_K$ has zero empirical error on all transitions from the K observed episodes. Thus,

$$\Pr\left[d_{\mathrm{compl},K}(\widehat{\mathcal{M}}_K, \mathcal{M}^*) > \epsilon\right] \leq |\mathcal{H}| \left(e^{-Km\epsilon} + K(m-1)\beta(k)\right).$$

Taking the complement proves that, with probability at least

$$1 - |\mathcal{H}| \left(e^{-Km\epsilon} + K(m-1)\beta(k)\right),$$

we have

$$d_{\text{compl},K}(\widehat{\mathcal{M}}_K, \mathcal{M}^*) \leq \epsilon.$$

Finally, to establish convergence in probability, choose $k = n$, so that $m = 1$. The mixing penalty then vanishes:

$$K(m-1)\beta(k) = 0.$$

Therefore, for every fixed $\epsilon > 0$,

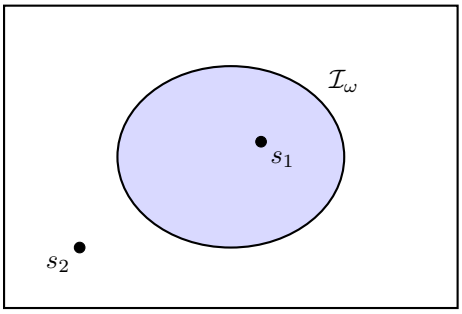
$$\Pr\left[d_{\text{compl},K}(\widehat{\mathcal{M}}_K, \mathcal{M}^*) > \epsilon\right] \leq |\mathcal{H}|e^{-K\epsilon} \longrightarrow 0 \qquad \text{as } K \to \infty.$$

Hence $d_{\text{compl},K}(\widehat{\mathcal{M}}_K, \mathcal{M}^*)$ converges to zero in probability. □

The β-mixing term captures temporal dependence among transitions within an episode. In SkillWrapper, the data-collection policy is updated only between episodes; conditioning on the preceding collection history therefore allows Lemma 2 to be applied separately within each episode, while still permitting adaptive exploration across episodes. The bound reflects the usual tradeoff between the effective sample size $m = \lfloor n/k \rfloor$ and the mixing penalty $(m-1)\beta(k)$. In particular, choosing $k = n$ yields $m = 1$, eliminating the within-episode mixing penalty and giving convergence as the number of collected episodes increases.

C.3 Illustration of Predicate Invention in Low-level Space

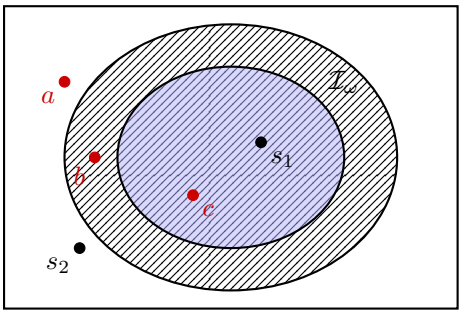
In this section, we illustrate how SkillWrapper correctly identifies circumstances where new predicates need to be invented. We here discuss cases for the precondition condition (Condition 1), and the effect condition follows the same logic.



Suppose that there exist two transitions: $\langle s_1, \underline{\omega}_1, s_1' \rangle, \langle s_2, \underline{\omega}_2, s_2' \rangle$ such that $s_1 \in \mathcal{I}_\omega, s_2 \notin \mathcal{I}_\omega$ initially. The learned model $\mathcal{M}$ so far is consistent with the two transitions.

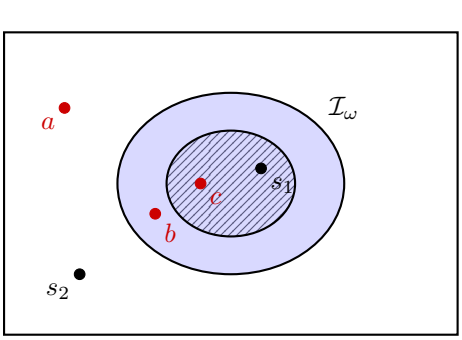
There are 3 possible circumstances of the estimated pre-success set Φ_ω resulting from $\mathcal{M}$ learned from $\langle s_1, \underline{\omega}_1, s_1' \rangle$ and $\langle s_2, \underline{\omega}_2, s_2' \rangle$. For each of them, we discuss all possible cases of other transitions with their initial state falling into different subsets of the state space (marked in red).

(1). The learned model has $\mathcal{I}_\omega \subset \Phi_\omega$. Then, for each subset that initial states of other transitions might fall in:



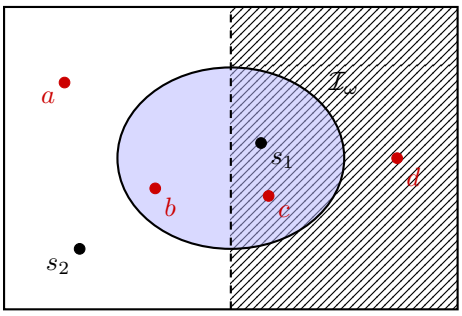
- **(a)**: $a \notin \Phi_\omega, a \notin \mathcal{I}_\omega$.
  $\nexists \langle s, \underline{\omega}, s' \rangle \in \mathcal{D}$ s.t. $s \in \Phi_\omega$ while $s \notin \mathcal{I}_\omega$.
  No need to invent new predicates.
- **(b)**: $b \in \Phi_\omega, b \notin \mathcal{I}_\omega$.
  $b \in \Phi_\omega, s_1 \in \Phi_\omega$, while $b \notin \mathcal{I}_\omega$ and $s_1 \in \mathcal{I}_\omega$. Condition 1 is satisfied, and a new predicate will be invented.
- **(c)**: $c \in \Phi_\omega, c \in \mathcal{I}_\omega$.
  $\nexists \langle s, \underline{\omega}, s' \rangle \in \mathcal{D}$ s.t. $s \in \Phi_\omega$ while $s \notin \mathcal{I}_\omega$.
  No need to invent new predicates.

(2). The learned model has $\Phi_\omega \subset \mathcal{I}_\omega$. Then, for each section that initial states of other transitions might fall in:



- **(a)**: $a \notin \Phi_\omega, a \notin \mathcal{I}_\omega$.
  $\nexists\, \langle s, \underline{\omega}, s' \rangle \in \mathcal{D}$ s.t. $s \in \Phi_\omega$ while $s \notin \mathcal{I}_\omega$.
  No need to invent new predicates.
- **(b)**: $b \notin \Phi_\omega, b \in \mathcal{I}_\omega$.
  This indicates the new transition is inconsistent with the learned model ($b \models \mathcal{I}_\omega$ but $b \not\models \Phi_\omega$). No new predicates need to be invented, and updating $\mathcal{M}$ by re-calculating the operators will ensure that every transition is supported. Formally, $\exists\, \underline{a}_\omega \in \underline{\mathcal{A}}_\omega$, such that $s \in \Phi_\omega, \forall \langle s, \omega, s' \rangle \in \mathcal{D}_\omega$. Then it is reduced to (a).
- **(c)**: $c \in \Phi_\omega, c \in \mathcal{I}_\omega$.
  $\nexists\, \langle s, \underline{\omega}, s' \rangle \in \mathcal{D}$ s.t. $s \in \Phi_\omega$ while $s \notin \mathcal{I}_\omega$.
  No need to invent new predicates.

(3). The learned model has $\Phi_\omega \cap \mathcal{I}_\omega \neq \emptyset$, $\Phi_\omega \not\subseteq \mathcal{I}_\omega$, $\mathcal{I}_\omega \not\subseteq \Phi_\omega$. Then, for each section that initial states of other transitions can fall in:



- **(a)**: $a \notin \Phi_\omega, a \notin \mathcal{I}_\omega$.
  $\nexists\, \langle s, \underline{\omega}, s' \rangle \in \mathcal{D}$ s.t. $s \in \Phi_\omega$ while $s \notin \mathcal{I}_\omega$.
  No need to invent new predicates.
- **(b)**: $b \notin \Phi_\omega, b \in \mathcal{I}_\omega$.
  Same situation as (2).b. No new predicates need to be invented, and updating $\mathcal{M}$ by re-calculating the operators will ensure that every transition is supported.
- **(c)**: $c \in \Phi_\omega, c \in \mathcal{I}_\omega$.
  $\nexists\, \langle s, \underline{\omega}, s' \rangle \in \mathcal{D}$ s.t. $s \in \Phi_\omega$ while $s \notin \mathcal{I}_\omega$.
  No need to invent new predicates.
- **(d)**: $d \in \Phi_\omega, d \notin \mathcal{I}_\omega$.
  $d \in \Phi_\omega, s_1 \in \Phi_\omega$, while $d \notin \mathcal{I}_\omega$ and $s_1 \in \mathcal{I}_\omega$. Condition 1 is satisfied, and a new predicate will be invented.

So far, we have discussed all possible cases of different transitions and Φ_ω resulting from the current model, and the predicate invention condition is proved to handle all cases.

C.4 Operator Learning

This section provides additional details on the operator learning algorithm. Specifically, it explains: **(Q1)** why Condition 1 uses learned operators as a measure for predicate invention without worrying about the operator learning algorithm being wrong, and **(Q2)** why it is impossible to have a transition predicted to be failed but actually successful when learning from a fixed dataset $\mathcal{D}$.

In this section, $\mathcal{D}$ denotes abstract transitions instead of low-level ones in the main paper, and $\mathcal{D}^{suc}$ and $\mathcal{D}^{fail}$ denote the subsets of successful and failed transitions in the dataset, respectively. Assuming preconditions are all conjunctive:

Proposition 2. *Let $\mathcal{A}_\omega$ be abstract actions learned by* SkillWrapper *from a subset of transitions $\mathcal{D}_\omega \subseteq \mathcal{D}$. For every successful transition, there exists an operator whose preconditions are satisfied by the initial state: Formally,*

$$\left(\langle s_i^{\uparrow}, \underline{a}, {s'}_i^{\uparrow} \rangle \in \mathcal{D}_\omega^{suc}\right) \implies \left(\exists\, \underline{a} \in \underline{\mathcal{A}}_\omega, s_i^{\uparrow} \models \textit{PRE}_{\underline{a}}\right).$$

Proof. Suppose each abstract action $a \in \mathcal{A}_\omega$ is learned from its corresponding subset of dataset $\boldsymbol{d}_\omega \subseteq \mathcal{D}_\omega$ sharing the same lifted abstract effects. SkillWrapper calculates its preconditions as:

$$\textsc{Pre}_a = \bigcap_{\langle s_i^\uparrow, \underline{a}, s'^\uparrow_i \rangle \in \mathcal{D}_\omega^{suc}} \Lambda(s_i^\uparrow)$$

By the definition of set intersection, the initial state of any successful transition must satisfy this intersection. Therefore, there must exist some $\underline{a} \in \underline{\mathcal{A}}_\omega$ such that $s_i^\uparrow \models \textsc{Pre}_{\underline{a}}$. □

Proposition 3. *Let $\mathcal{A}_\omega$ be abstract actions learned by* SkillWrapper *from a subset of transitions $\mathcal{D}_\omega \subseteq \mathcal{D}$. If none of the operators in $\mathcal{A}_\omega$ has its preconditions satisfied by the initial state of a transition, the transition must fail. Formally,*

$$\big(\forall \underline{a} \in \underline{\mathcal{A}}_\omega, s_i \not\models \textit{Pre}_{\underline{a}}\big) \implies \big(\langle s_i^\uparrow, \underline{a}, s'^\uparrow_i \rangle \in \mathcal{D}_\omega^{fail}\big).$$

Proof. Suppose, for each operator $a \in \mathcal{A}_\omega$ learned from its corresponding subset of dataset $\boldsymbol{d}_\omega \subseteq \mathcal{D}_\omega$ with the same abstract effects. For any successful initial state, $\Lambda(s^\uparrow) \models \big(\bigcap_{\langle s_i^\uparrow, \underline{a}, s'^\uparrow_i \rangle \in \mathcal{D}_\omega^{suc}} \Lambda(s^\uparrow{}_i)\big) = \textsc{Pre}_a$, because taking intersection only relax the precondition. Hence, for any state $s^\uparrow \notin \textsc{Pre}_a$, it would mean that its corresponding transition is not included in $\mathcal{D}_\omega^{suc}$. Given that any transition in $\mathcal{D}_\omega$ must be either successful or failed, $\langle s_i^\uparrow, \underline{a}, s'^\uparrow_i \rangle \in \mathcal{D}_\omega^{fail}$. □

(A1) With the above properties established, we have narrowed down the only unproven case to false negatives—where $\langle s_i^\uparrow, \underline{a}, s'^\uparrow_i \rangle \in \mathcal{D}_\omega^{fail}$ but $\exists\, \underline{a} \in \underline{\mathcal{A}}, s_i^\uparrow \models \textsc{Pre}_{\underline{a}}$. In such a case, further refinement is infeasible with the current abstract state representations unchanged, and predicate invention must be triggered to distinguish these false negatives.

(A2) The scenario where a transition's initial state fails to satisfy the preconditions of any $a \in \mathcal{A}$ yet still executes successfully is impossible under a fixed $\mathcal{D}$. Such a transition would necessarily belong to $\mathcal{D}^{suc}$, a contradiction directly ruled out by Proposition 2.

D Additional Details on the Hypothesis Class and Sample Complexity

Recall that our learned symbolic model has the form

$$\mathcal{M} = (\mathcal{P}, \mathcal{A}),$$

where $\mathcal{P}$ is a set of predicates and $\mathcal{A} = \bigcup_{\omega \in \Omega} \mathcal{A}_\omega$ is a set of abstract operators, with $\mathcal{A}_\omega$ the set of operators associated with skill $\omega \in \Omega$.

Each operator $a \in \mathcal{A}_\omega$ is defined by its preconditions and add/delete effects:

$$a \triangleq \big(\omega, \boldsymbol{\theta}_a, \textsc{Pre}_a, \textsc{Eff}_a^+, \textsc{Eff}_a^-\big), \quad \textsc{Pre}_a, \textsc{Eff}_a^+, \textsc{Eff}_a^- \subseteq \mathcal{P}.$$

In our implementation, the number of VLM calls is finite, and the resulting models use a small number of predicates in environments with finitely many objects. For the *theoretical analysis*, we make this implicit resource bound explicit:

- Let $P_{\max}$ denote a fixed maximum number of predicates that SkillWrapper is allowed to invent.
- Let $\mathcal{P}_{\mathrm{FM}}$ denote the finite family of predicate classifiers expressible by the FM.
- Let $\mu_{\max}$ denote the maximum arity of any predicate $\sigma \in \mathcal{P}$.

Because SkillWrapper learns one operator per lifted effect cluster, we can derive an upper bound on the number of operators per skill $\omega \in \Omega$ based on the number of possible effect sets. For any operator $a \in \mathcal{A}_\omega$ and predicate $\sigma \in \mathcal{P}$, there are three possible cases: $\sigma \in \textsc{Eff}_a^+$, $\sigma \in \textsc{Eff}_a^-$, or $\sigma \notin \textsc{Eff}_a^- \cup \textsc{Eff}_a^+$. Because the upper bound of possible instances of p is $|\mathcal{O}|^{\mu_{\max}}$, we can express the maximum number of operators as

$$A_{\max} = 3^{P_{\max} \cdot |\mathcal{O}|^{\mu_{\max}}}$$

We therefore define the hypothesis class analyzed in Theorem 2 as

$$\mathcal{H} = \Big\{ (\mathcal{P}, \mathcal{A}) \mid \mathcal{P} \subseteq \mathcal{P}_{\mathrm{FM}},\ |\mathcal{P}| \le P_{\max},\ |\mathcal{A}_\omega| \le A_{\max}\, \forall \omega \in \Omega \Big\}. \tag{6}$$

D.1 Upper Bound on $|\mathcal{H}|$

We now derive a practical upper bound on the size of $\mathcal{H}$ in equation 6.

Let $\mathcal{P}_{\mathrm{FM}}$ denote the finite family of predicate classifiers expressible by the FM. Since a learned model contains at most $P_{\max}$ predicates, the number of possible predicate sets $\mathcal{P} \subseteq \mathcal{P}_{\mathrm{FM}}$ is bounded by

$$\sum_{j=0}^{P_{\max}} \binom{|\mathcal{P}_{\mathrm{FM}}|}{j} \;\le\; \left(|\mathcal{P}_{\mathrm{FM}}|+1\right)^{P_{\max}}.$$

Fix one such predicate set $\mathcal{P}$ with $|\mathcal{P}| \le P_{\max}$. For each operator α, its symbolic definition is given by three subsets of $\mathcal{P}$:

$$\textsc{Pre}_a,\ \textsc{Eff}_a^+,\ \textsc{Eff}_a^- \subseteq \mathcal{P}.$$

We consider negative preconditions, so there are three possibilities for one predicate p: $p \in \textsc{Pre}_a$, $-p \in \textsc{Pre}_a$, and $p \notin \textsc{Pre}_a$. Since $\textsc{Eff}_a^+$ and $\textsc{Eff}_a^-$ are considered in $A_{\max}$, a single operator has at most

$$3^{P_{\max}\cdot|\mathcal{O}|^{\mu_{\max}}} \tag{7}$$

distinct configurations of preconditions.

For a fixed skill $\omega \in \Omega$, we allow at most $A_{\max}$ operators. Treating each of the $A_{\max}$ operator "slots" as independently choosing one of the $3^{P_{\max}|\mathcal{O}|^{\mu_{\max}}}$ possible configurations in equation 7, the total number of operator-sets $\mathcal{A}_\omega$ for that skill is bounded by

$$\left(3^{P_{\max}\cdot|\mathcal{O}|^{\mu_{\max}}}\right)^{A_{\max}} \;=\; 3^{P_{\max}A_{\max}|\mathcal{O}|^{\mu_{\max}}}. \tag{8}$$

Across all skills $\omega \in \Omega$ and all possible predicate sets, we obtain the bound

$$|\mathcal{H}| \;\le\; \left(|\mathcal{P}_{\mathrm{FM}}|+1\right)^{P_{\max}} \prod_{\omega\in\Omega} 3^{P_{\max}A_{\max}|\mathcal{O}|^{\mu_{\max}}} \;=\; \left(|\mathcal{P}_{\mathrm{FM}}|+1\right)^{P_{\max}} 3^{P_{\max}A_{\max}|\Omega||\mathcal{O}|^{\mu_{\max}}}. \tag{9}$$

D.2 Sample Complexity for a Target (ϵ, δ)

Theorem 2 states that, for any $\epsilon > 0$,

$$\Pr[d_{\mathrm{compl},K}(\widehat{\mathcal{M}}_K, \mathcal{M}^*) > \epsilon] \le |\mathcal{H}|\left(e^{-Km\epsilon} + K(m-1)\beta(k)\right), \tag{10}$$

where $m = \lfloor n/k \rfloor$ is the effective sample size, $\widehat{\mathcal{M}}_K$ is the model returned by SkillWrapper after observing K episodes of n transitions each, and d_{compl} is the distance defined in the main text (probability of a "missed feasible" event under the transition distribution).

Substituting the bound on $|\mathcal{H}|$ from equation 9 into equation 10 yields

$$\begin{aligned}\Pr\left[d_{\mathrm{compl},K}\left(\widehat{\mathcal{M}}_K, \mathcal{M}^*\right) > \epsilon\right] \;\le\; &\left(|\mathcal{P}_{\mathrm{FM}}|+1\right)^{P_{\max}} 3^{P_{\max}A_{\max}|\Omega||\mathcal{O}|^{\mu_{\max}}} \\ &\cdot \left(e^{-Km\epsilon} + K(m-1)\beta(k)\right).\end{aligned} \tag{11}$$

Let $C = \left(|\mathcal{P}_{\mathrm{FM}}|+1\right)^{P_{\max}} 3^{P_{\max}A_{\max}|\Omega||\mathcal{O}|^{\mu_{\max}}}$. To guarantee that this probability is at most $\delta \in (0,1)$, it suffices that

$$C\left(e^{-Km\epsilon} + K(m-1)\beta(k)\right) \;\le\; \delta.$$

Instead of trying to solve for n simultaneously across both terms, we split the total allowable failure probability (δ) in half. To guarantee the total sum is less than δ, it suffices to ensure that each individual piece is less than $\delta/2$:

$$Ce^{-Km\epsilon} \le \frac{\delta}{2}, \tag{12}$$

$$CK(m-1)\beta(k) \le \frac{\delta}{2} \tag{13}$$

Solve equation 12, we get

$$m \ge \frac{1}{K\epsilon}\left(\ln C + \ln 2 + \ln\frac{1}{\delta}\right)$$

Substitute back C:

$$m \geq \frac{1}{K\epsilon}\left(P_{\max}\ln(|\mathcal{P}_{\mathrm{FM}}|+1) + P_{\max}A_{\max}|\Omega||\mathcal{O}|^{\mu_{\max}}\ln 3 + \ln 2 + \ln\frac{1}{\delta}\right) \tag{14}$$

Now we look at the second split inequality in equation 13:

$$\beta(k) \leq \frac{\delta}{2CK(m-1)}$$

In β-mixing process, the mixing coefficient $\beta(k)$ typically decays exponentially with the time-gap parameter k (i.e., $\beta(k) \leq c \cdot e^{-bk}$ for physical constants $c, b > 0$). Apply this standard assumption, equation 13 becomes

$$c \cdot e^{-bk} \leq \frac{\delta}{2CK(m-1)}.$$

Solve the equation, we have

$$k \geq \frac{1}{b}\left(\ln C + \ln K + \ln(m-1) + \ln\frac{1}{\delta} + \ln(2c)\right)$$

Using the error-splitting technique under standard exponential mixing ($\beta(k) \leq c \cdot e^{-bk}$), we found:

$$m = \mathcal{O}\left(\frac{\ln C + \ln(1/\delta)}{K\epsilon}\right) \tag{15}$$

$$k = \mathcal{O}\left(\ln C + \ln K + \ln(1/\delta) + \ln(m)\right) \tag{16}$$

To simplify the algebra, let $V = \ln C + \ln(1/\delta)$. This yields

$$m = \mathcal{O}\left(\frac{V}{K\epsilon}\right) \quad \text{and} \quad k = \mathcal{O}\left(V + \ln(Km)\right)$$

Since $Km = O(V/\epsilon)$,

$$k = \mathcal{O}\left(V + \ln\left(\frac{V}{\epsilon}\right)\right) = \mathcal{O}\left(V + \ln(V) + \ln\left(\frac{1}{\epsilon}\right)\right).$$

Because V grows faster than its own logarithm ($\ln(V)$), the term $\ln(V)$ gets absorbed by V term. This leaves a clean bound for k:

$$k = \mathcal{O}\left(V + \ln\left(\frac{1}{\epsilon}\right)\right) = \mathcal{O}\left(\ln C + \ln\left(\frac{1}{\delta}\right) + \ln\left(\frac{1}{\epsilon}\right)\right)$$

Since $m = \lfloor n/k \rfloor$, per-episode sample complexity satisfies $mk \leq n < (m+1)k$. Multiplying the two complexities:

$$n = \mathcal{O}\left(\left[\frac{\ln C + \ln(1/\delta)}{K\epsilon}\right] \cdot \left[\ln C + \ln\left(\frac{1}{\delta}\right) + \ln\left(\frac{1}{\epsilon}\right)\right]\right),$$

and the total sample complexity:

$$N = Kn = \mathcal{O}\left(\left[\frac{\ln C + \ln(1/\delta)}{\epsilon}\right] \cdot \left[\ln C + \ln\left(\frac{1}{\delta}\right) + \ln\left(\frac{1}{\epsilon}\right)\right]\right).$$

The final total sample complexity of N to ensure

$$d_{\mathrm{compl,K}}\left(\widehat{\mathcal{M}}_K, \mathcal{M}^*\right) \leq \epsilon \quad \text{with probability at least } 1-\delta$$

is expressed as

$$N = \mathcal{O}\left(\frac{1}{\epsilon}\Big[\left(P_{\max}\ln\left(|\mathcal{P}_{\mathrm{FM}}|+1\right)+P_{\max}A_{\max}|\Omega||\mathcal{O}|^{\mu_{\max}}+\ln(1/\delta)\right)^2 + \left(P_{\max}\ln\left(|\mathcal{P}_{\mathrm{FM}}|+1\right)+P_{\max}A_{\max}|\Omega||\mathcal{O}|^{\mu_{\max}}+\ln(1/\delta)\right)\ln(1/\epsilon)\Big]\right). \tag{17}$$

In our experiments, the arity and realized numbers of predicates are small. Setting $P_{\max}$ to match the practical budget yields numerical values in equation 17 for the regimes we study. Our theoretical result therefore formalizes how increasing the capacity of the symbolic model (via larger $P_{\max}$) trades off against the number of transitions needed to achieve a desired completeness level.

E Comparisons of Symbolic Model Learning Methods

Table 3: Comparison of symbolic model learning methods.

Method	Fully Obs. & Deterministic	No Initial Pred.	Theoretical Analysis	Auton. Learning	Interp. Rep.	Real-World Eval.
Liang et al. (2025)	✓				✓	
Athalye et al. (2026)	✓				✓	✓
Liang et al. (2026)	✓				✓	
Silver et al. (2021)	✓					
Silver et al. (2023)	✓					
Han et al. (2024)	✓	✓			✓	✓
Konidaris et al. (2018)		✓	✓			✓
Shah et al. (2025)	✓	✓				✓
SkillWrapper **(Ours)**	✓	✓	✓	✓	✓	✓

Fully Observable and Deterministic Assumption Prior and concurrent work in predicate invention for model learning and symbolic model learning commonly make two key assumptions, deterministic skills and a fully observable environment; SkillWrapper follows this paradigm. In the future, to relax these assumptions, modeling states as distributions instead of sets and introducing probabilistic representations—such as Probabilistic PDDL (PPDDL)—presents a promising direction, if they are grounded in a concrete theoretical framework. A prominent example of work that relaxes these assumptions is provided by Konidaris et al. (2018).

Predicate Learning from Scratch The problem of predicate invention can be categorized into "completing partial operators" and "learning from scratch". In the former setting (Liang et al., 2025; 2026; Athalye et al., 2026; Silver et al., 2023; 2021), the system is initialized with predefined goal predicates that partially specify the operators. Because these methods do not invent predicates needed to model skill effects, the predefined goal predicates must already be assigned to the appropriate locations in the partial operators. Moreover, predefined tasks are provided at the same time to enable the collection of transition data. Together, these requirements impose substantial prior knowledge. In the latter setting, the system must begin with an empty predicate set. This "learning from scratch" approach is significantly more challenging but better reflects realistic scenarios where untrained users, rather than planning experts, interact with the system. SkillWrapper adopts this latter paradigm, aligning with the work of Konidaris et al. (2018); Shah et al. (2025); Han et al. (2024).

Theoretical Analysis Theoretical rigor is essential for learning a task-level model with formal guarantees. While existing research has explored predicate invention through planning heuristics (Silver et al., 2023; 2021) or derived guarantees for non-generative cases (Konidaris et al., 2018), we contend that current generative predicate invention approaches (Liang et al., 2025; 2026; Athalye et al., 2026) do not characterize the conditions under which the learned abstractions support sound and complete planning. This limitation primarily stems from their reliance on greedy best-first search and predefined goal predicates. In contrast, SkillWrapper only invents predicates when necessary to reconcile the symbolic model with observed data, and validates that a proposed predicate improves the model before keeping it. Hence, our method does not require an ad-hoc greedy predicate selection procedure, enabling a formal analysis of when these properties hold, as shown in Appendix C.

Autonomous Learning As demonstrated in our experiments, efficiently collecting transition data within object-centric environments with parameterized skills remains a significant challenge. Current approaches typically rely on pre-collected datasets (Konidaris et al., 2018; Shah et al., 2025), human guidance (Han et al., 2024), or the specification of explicit training tasks (Liang et al., 2025; 2026; Athalye et al., 2026; Silver et al., 2021; 2023). In contrast, achieving true scalability requires a shift toward autonomous learning. SkillWrapper introduces a novel paradigm for this purpose, utilizing goal-agnostic skill sequences for pure exploration. By decoupling data collection from specific tasks or human guidance, SkillWrapper offers a more flexible and promising path forward.

Interpretable Representation In previous works where abstractions are captured by in-domain classifiers (Konidaris et al., 2018) or generated via fixed templates (Silver et al., 2021; 2023), specifying real-world tasks was challenging because the predicates and operators were unreadable by non-experts, and often required manual inspection of the states where classifiers were triggered or templates were satisfied. In contrast, recent research leveraging LLMs or VLMs for predicate invention produces abstractions that are directly human-interpretable (Liang et al., 2025; 2026; Athalye et al., 2026). Within this context, SkillWrapper is the first to characterize the essential properties required for such systems.

Real-World Evaluation Evaluation on physical robotic platforms is essential to demonstrate a system's practical applicability. Traditional Task and Motion Planning (TAMP) typically assumes fully observable and deterministic environments, effectively abstracting away sensory noise within its theoretical frameworks. Therefore, real-world evaluation was not necessary since the environments are perfectly controlled, assuming full access to perfect state information, such as object poses. However, recent works leveraging foundation models without real-world evaluation (Liang et al., 2025; 2026) have raised concerns regarding the noise and uncertainty inherent in probabilistic neural networks. To address these concerns, we highlight the reliability of SkillWrapper in real-world scenarios by providing a detailed evaluation of its performance in inventing and classifying predicates under the complexities of physical interaction (see Appendix H).

F Operator Learning Details

F.1 Lifting and Substitution

Object lifting removes object identities from an abstract state while preserving their types and relational structure, allowing predicate invention and operator learning to compare transitions involving different object instances. Given an abstract state $s^\uparrow$ and object tuple $\boldsymbol{o}$, let $s^\uparrow|_{\boldsymbol{o}}$ denote the restriction of $s^\uparrow$ to grounded predicates containing at least one object in $\boldsymbol{o}$. For a substitution $\theta = [\boldsymbol{v}/\boldsymbol{o}]$, let $\theta(f)$ denote the result of replacing each object o_i in a grounded predicate formula f with the corresponding typed variable v_i.

In SkillWrapper, the substitution associated with $\boldsymbol{o}$ is constructed deterministically from the skill arguments and object types. We therefore define the lifting function as

$$\Lambda_{\boldsymbol{o}}(s^\uparrow) \coloneqq \theta_{\boldsymbol{o}}\left(s^\uparrow|_{\boldsymbol{o}}\right),$$

where $\theta_{\boldsymbol{o}}$ is the deterministic substitution induced by $\boldsymbol{o}$. Thus, two low-level states s_i and s_j, associated with object tuples $\boldsymbol{o}_i$ and $\boldsymbol{o}_j$, have the same lifted abstract representation when

$$\Lambda_{\boldsymbol{o}_i}(\alpha(s_i)) = \Lambda_{\boldsymbol{o}_j}(\alpha(s_j)).$$

This allows transitions involving different object identities to be clustered and compared in a common lifted representation.

F.2 Running Example

We provide a running example here to illustrate the operator learning algorithm. Suppose that the environment contains the following objects with their typing: `Robot` $-$ [`robot`], `Table` $-$ [`table`], `Bottle` $-$ [`object`, `openable`], `Apple` $-$ [`object`], and `Banana` $-$ [`object`]. The environment has these predicates:

- (`is_on` ?`object` ?`table`): The object is on the table.
- (`hand_free` ?`robot`): The robot has a free hand or gripper.
- (`is_holding` ?`robot` ?`object`): The robot is currently holding the object.

We have collected three successful transitions for the skill `PickUp`(`robot`, `object`):

```
Transition 1:
   Skill: PickUp(Robot, Bottle)
```

```
   Initial State: (and
      (is_on Bottle Table)
      (hand_free Robot)
      (not (is_holding Robot Bottle))
      (is_on Banana Table)
      )

   Goal State: (and
      (not (is_on Bottle Table))
      (not (hand_free Robot))
      (is_holding Robot Bottle)
      (is_on Banana Table)
      )

Transition 2:
   Skill: PickUp(Robot, Apple)
   Initial State: (and
      (is_on Apple Table)
      (hand_free Robot)
      (not (is_holding Robot Apple))
      (is_on Banana Table)
      )

   Goal State: (and
      (not (is_on Apple Table))
      (not (hand_free Robot))
      (is_holding Robot Apple)
      (is_on Banana Table)
      )

Transition 3:
   Skill: PickUp(Robot, Bottle)
   Initial State: (and
      (hand_free Robot)
      (not (is_holding Robot Bottle))
      (is_on Banana Table)
      )

   Goal State: (and
      (not (hand_free Robot))
      (is_holding Robot Bottle)
      (is_on Banana Table)
      )
```

We first lift all objects with respect to the skill instances and calculate the lifted effect, so that skills operating on different objects can be grouped into one cluster.

```
Transition 1:
   Eff+: < (is_holding ?robot_1 ?object_1 ) >
   Eff-: < (is_on ?object_1 ?table_1 ), (hand_free ?robot_1) >

Transition 2:
   Eff+: < (is_holding ?robot_1 ?object_1 ) >
   Eff-: < (is_on ?object_1 ?table_1 ), (hand_free ?robot_1) >

Transition 3:
   Eff+: < (is_holding ?robot_1 ?object_1 ) >
   Eff-: < (hand_free ?robot_1) >
```

Then, transition 1 and 2 fall in the same cluster because they share the same effect, whereas transition 3 forms a cluster itself. From the two clusters, we can calculate their preconditions by taking the intersections of their initial states in the lifted form.

```
Cluster 1 (Transition 1, 2):
   Pre: (and
   (not (is_holding ?robot_1 ?object_1))
   (is_on ?object_1 ?table_1)
   (hand_free ?robot_1)
   )

Cluster 2 (Transition 3):
   Pre: (and
   (not (is_holding ?robot_1 ?object_1))
   (hand_free ?robot_1)
   )
```

Their action parameters are determined by taking the union of the parameters in their precondition and effects. Finally, we learn two operators for the skill `PickUp(robot, object)`:

```
Operator 1 (Transition 1,2):
(:action Pick_1
:parameters ( ?robot_1 - robot ?object_1 - object ?table_1 - table)
:precondition (and
   (not (is_holding ?robot_1 ?object_1))
   (hand_free ?robot_1 )
   (is_on ?object_1 ?table_1)
)
:effect (and
   (is_holding ?robot_1 ?object_1 )
   (not (is_on ?object_1 ?table_1 ))
   (not (hand_free ?robot_1))
 )
)

Operator 2 (Transition 3):
(:action Pick_2
:parameters ( ?robot_1 - robot ?openable_1 - openable )
:precondition (and
   (not (is_holding ?robot_1 ?openable_1))
   (hand_free ?robot_1 )
)
:effect (and
   (is_holding ?robot_1 ?openable_1 )
   (not (hand_free ?robot_1))
 )
)
```

We highlight that the operator learning algorithm of SkillWrapper is general and practical with these important details:

- SkillWrapper's operator learning algorithm can capture conditional effects, provided that the relevant predicates are nullary or involve at least one skill parameter, such as `is_on(?object, ?table)`, in the relevant grounded abstract transitions.
- SkillWrapper's operator learning algorithm handles multi-type objects by using the lowest level of the type hierarchy consistent with the data, preventing over-generalization while retaining compositional structure. In the example above, we learn the parameter of `PickUp_2` to be `?openable` instead of `?object`.
- SkillWrapper's operator learning algorithm clusters transitions after lifting them, which enables efficient symbolic model learning. In the example above, we successfully learn from cluster 1, where transitions belong to different grounded skills.
- SkillWrapper's operator learning algorithm discards tautological predicates—those that are always true or always false. In the example, `(is_on Banana Table)` is true for all states, and thus is removed at the initial step.

G Case Studies, Learned Operators, and Example Tasks

G.1 Result Analysis of Baseline Comparisons

We present an analysis of the failure modes observed in SkillWrapper and its baselines; note that the System Predicates baseline is evaluated exclusively within the Robotouille domain. One common failure mode is misclassifications during inference time, specifically when abstracting raw visual observations into symbolic states using the learned predicates. As our primary contributions are centered on theoretical and algorithmic design, we did not incorporate advanced prompting or visual-reasoning techniques—such as ViperGPT (Surís et al., 2023)—which could potentially mitigate these perception-level errors.

Expert Operators function as oracle knowledge and represent the theoretical upper bound for learning-based systems, as evidenced by the results in the Robotouille (Gonzalez-Pumariega et al., 2025) domain. However, this baseline's performance degrades significantly in real-world experiments, where it is outperformed by SkillWrapper and other learning-based methods. A potential explanation for this shift is that foundation models may more effectively ground and interpret predicate semantics when those abstractions are self-generated rather than manually defined by human experts. The central advantage of our approach is that it reduces reliance on domain expertise, empowering robots to autonomously derive symbolic representations that are inherently tailored to their specific embodiment, physical capabilities, and task distributions.

System Predicates utilizes a fixed set of eight predicates designed to cover all possible states within the domain. However, this compact representation proves insufficient for capturing essential environmental dynamics. Specifically, the absence of a predicate `clear_above` (or its semantic equivalent) prevents the model from representing the critical precondition of the *Pick* skill: that an object must be unobstructed from the top to be successfully grasped. Because predicate invention is disabled in this baseline, the planner frequently generates unsound plans—such as attempting to retrieve an object from the bottom of a stack—resulting in a significantly lower success rate for high-complexity or impossible tasks.

ViLa's failure modes include neglecting skill parameter constraints, repeating failed skills within its action history, and failing to recognize goal state has been reached. Although SkillWrapper still outperformed this baseline, ViLa exhibits stronger-than-expected results in the Bi-manual Kuka domain, likely due to similarities with its training distribution. A critical finding is its consistently poor performance on *Impossible* tasks, which underscores the risks of unsoundness and the reliability concerns typical of purely generative planners.

Random Exploration is severely constrained by the inherent inefficiency of stochastic data collection. Despite this, the predicates and operators it manages to acquire are of high quality, as this baseline utilizes the same underlying invention algorithm as SkillWrapper. The primary failure mode stems from the low probability of randomly executing skills with complex preconditions. This hinders the invention process, because predicate invention can only happen when the skills are successfully executed. For example, in the Robotouille setting, the *Cook* action requires the item to be on the stove and the robot's hand to be empty. Consequently, it yields near-zero success rates across most domains, solving only the simplest tasks where basic skills like *Pick* suffice. Furthermore, its good performance on *Impossible* tasks is a byproduct of model incompleteness: because it lacks the operators necessary to chain complex actions together, the planner correctly—albeit for the wrong reasons—identifies those tasks as unsolvable.

FM Invent tends to continuously propose new predicates regardless of whether the existing set is sufficiently expressive to model the domain. While it successfully gathers transition data for all skills by leveraging the active collection algorithm of SkillWrapper, the over-invented predicates impose a significant classification burden on the foundation model during inference. Consequently, the learned operators become over-specified, containing numerous redundant preconditions and effects that hinder effective planning. This explains the baseline's poor success rate alongside its high performance on *Impossible* tasks; the resulting model is so overly constrained by superfluous symbolic logic that it essentially views most valid transitions as infeasible.

No Heuristic exhibits a failure mode similar to System Predicates, e.g., performing well on *Easy* problems but degrading significantly on *Hard* tasks in Robotouille domain. Our investigation reveals

that this baseline generally invents fewer predicates than SkillWrapper, leading to the omission of critical symbolic logic in two out of five experimental runs. This explains the high variance observed in *Hard* and *Impossible* scenarios and suggests that *Easy* tasks can often be solved even with an incomplete predicate set—a finding that mirrors our observations of System Predicates in the Robotouille domain. Taken together with the Franka domain results, these findings validate the utility of our two engineered heuristics for skill sequence proposal. Interestingly, performance in the bi-manual Kuka experiments remained relatively stable; we attribute this to transition "dead-ends" that naturally constrain the exploration space, combined with preconditions that are simpler to model than those in other domains. Nevertheless, enhancing exploration strategies remains a promising avenue for future research. Finally, we note that while metrics such as success rate and planning budget effectively evaluate learned operators, there is a distinct need for novel metrics specifically designed to assess exploration efficiency.

SkillWrapper's failure mode is similar to the case of Random Exploration. In the Robotouille domain, for instance, a comparison between the PDDL operators learned by SkillWrapper and those of the Expert baseline reveals that the inclusion of just one extra predicate, `on_cutting_board(item)`, partitions the transition data into excessively small clusters. Consequently, the transition data in these smaller clusters cannot support the model learning algorithm to effectively filter out spurious preconditions. This case study underscores the critical equilibrium required between predicate invention and data acquisition: if a system over-invents predicates relative to available transition data, the resulting operators become over-specified and fail to generalize. In SkillWrapper, this balance is empirically regulated via the scoring function threshold and the length of the proposed skill sequences. A remaining sub-optimality is that our current framework does not filter for semantic synonyms or antonyms, which increases the classification burden on the foundation model. While these redundancies are typically managed by the model's internal reasoning, a more sophisticated prompting system could further mitigate this issue and enhance computational efficiency.

G.2 Learned Predicates & Operators

```
# Learned Predicates & Operators of Burger domain

# Predicates
 name: hand_empty
 types:
 - robot
 semantic: the robot isn't holding any pickupable object.

 name: station_surface_empty
 types:
 - station
 semantic: no pickupable object is resting on the station's top surface.

 name: chopped
 types:
 - cuttable
 semantic: the cuttable object is visually in multiple small, separated pieces rather than a
 single intact piece.

 name: holding
 types:
 - robot
 semantic: the robot's gripper is visibly grasping and carrying the specified pickupable
 object.

 name: cooked
 types:
 - cookable
 semantic: the specified cookable exhibits a cooked appearance (browned surface or grill
 marks, not raw/pink).

 name: on_station
 types:
 - pickupable
 - station
 semantic: the specified pickupable is visibly resting in contact with the station's top
 surface.
```

```
  name: stacked_on
  types:
  - pickupable
  - pickupable
  semantic: the first pickupable is visibly in direct contact and vertically supported by the
  second pickupable, with no other object between them.

  name: top_surface_clear
  types:
  - pickupable
  semantic: no pickupable object is in direct vertical contact resting on the specified
  pickupable.

  name: on_stove_surface
  types:
  - cookable
  semantic: the specified cookable is visibly in direct contact with and supported by the
  stove's cooking surface (grill area), not held by any agent or resting on another object.

# Operators
(:action Pick_1
:parameters ( ?pickupable_p3 - pickupable ?pickupable_p5 - pickupable ?pickupable_p6 -
pickupable ?robot_p0 - robot ?station_p1 - station ?station_p2 - station )
:precondition (and
      (hand_empty ?robot_p0)
      (on_station ?pickupable_p6 ?station_p1)
      (top_surface_clear ?pickupable_p6)
      (not (holding ?robot_p0 ?pickupable_p3))
      (not (holding ?robot_p0 ?pickupable_p5))
      (not (holding ?robot_p0 ?pickupable_p6))
      (not (on_station ?pickupable_p6 ?station_p2))
      (not (station_surface_empty ?station_p1))
)
:effect (and
      (holding ?robot_p0 ?pickupable_p6)
      (station_surface_empty ?station_p1)
      (not (hand_empty ?robot_p0 ))
      (not (on_station ?pickupable_p6 ?station_p1))
    )
)

(:action Pick_2
:parameters ( ?pickupable_p6 - pickupable ?robot_p0 - robot )
:precondition (and
      (hand_empty ?robot_p0 )
      (top_surface_clear ?pickupable_p6 )
      (not (holding ?robot_p0 ?pickupable_p6))
)
:effect (and
      (holding ?robot_p0 ?pickupable_p6)
      (not (hand_empty ?robot_p0))
 )
)

(:action Pick_3
:parameters ( ?pickupable_p3 - pickupable ?pickupable_p4 - pickupable ?pickupable_p6 -
pickupable ?robot_p0 - robot )
:precondition (and
      (hand_empty ?robot_p0 )
      (stacked_on ?pickupable_p4 ?pickupable_p3)
      (top_surface_clear ?pickupable_p4 )
      (not (holding ?robot_p0 ?pickupable_p3))
      (not (holding ?robot_p0 ?pickupable_p4))
      (not (holding ?robot_p0 ?pickupable_p6))
      (not (top_surface_clear ?pickupable_p3))
)
:effect (and
      (holding ?robot_p0 ?pickupable_p4)
      (top_surface_clear ?pickupable_p3 )
      (not (hand_empty ?robot_p0 ))
      (not (stacked_on ?pickupable_p4 ?pickupable_p3))
 )
)

(:action Place_1
:parameters ( ?pickupable_p3 - pickupable ?pickupable_p4 - pickupable ?pickupable_p5 -
pickupable ?pickupable_p6 - pickupable ?robot_p0 - robot ?station_p2 - station )
:precondition (and
      (holding ?robot_p0 ?pickupable_p4)
      (station_surface_empty ?station_p2 )
      (top_surface_clear ?pickupable_p4 )
```

```
      (not (hand_empty ?robot_p0 ))
      (not (holding ?robot_p0 ?pickupable_p5))
      (not (holding ?robot_p0 ?pickupable_p6))
      (not (on_station ?pickupable_p4 ?station_p2))
      (not (stacked_on ?pickupable_p3 ?pickupable_p4))
      (not (stacked_on ?pickupable_p4 ?pickupable_p3))
)
:effect (and
      (hand_empty ?robot_p0 )
      (on_station ?pickupable_p4 ?station_p2)
      (not (holding ?robot_p0 ?pickupable_p4))
      (not (station_surface_empty ?station_p2 ))
 )
)

(:action Place_2
:parameters ( ?pickupable_p5 - cookable ?pickupable_p3 - pickupable ?pickupable_p4 -
pickupable ?pickupable_p6 - pickupable ?robot_p0 - robot ?station_p1 - station )
:precondition (and
      (holding ?robot_p0 ?pickupable_p5)
      (station_surface_empty ?station_p1)
      (top_surface_clear ?pickupable_p5)
      (not (hand_empty ?robot_p0))
      (not (holding ?robot_p0 ?pickupable_p3))
      (not (holding ?robot_p0 ?pickupable_p4))
      (not (holding ?robot_p0 ?pickupable_p6))
      (not (on_station ?pickupable_p3 ?station_p1))
      (not (on_station ?pickupable_p5 ?station_p1))
      (not (on_stove_surface ?pickupable_p5))
      (not (stacked_on ?pickupable_p5 ?pickupable_p6))
      (not (stacked_on ?pickupable_p6 ?pickupable_p5))
)
:effect (and
      (hand_empty ?robot_p0 )
      (on_station ?pickupable_p5 ?station_p1)
      (on_stove_surface ?pickupable_p5 )
      (not (holding ?robot_p0 ?pickupable_p5))
      (not (station_surface_empty ?station_p1 ))
 )
)

(:action Stack_1
:parameters ( ?pickupable_p3 - pickupable ?pickupable_p4 - pickupable ?pickupable_p5 -
pickupable ?robot_p0 - robot )
:precondition (and
      (holding ?robot_p0 ?pickupable_p3)
      (top_surface_clear ?pickupable_p3 )
      (top_surface_clear ?pickupable_p4 )
      (not (hand_empty ?robot_p0 ))
      (not (holding ?robot_p0 ?pickupable_p4))
      (not (holding ?robot_p0 ?pickupable_p5))
      (not (stacked_on ?pickupable_p3 ?pickupable_p4))
   )
:effect (and
      (hand_empty ?robot_p0 )
      (stacked_on ?pickupable_p3 ?pickupable_p4)
      (not (holding ?robot_p0 ?pickupable_p3))
      (not (top_surface_clear ?pickupable_p4 ))
 )
)

(:action Cut_1
:parameters ( ?cuttable_p4 - cuttable ?pickupable_p3 - pickupable ?pickupable_p6 - pickupable
?robot_p0 - robot ?station_p2 - station )
:precondition (and
      (hand_empty ?robot_p0 )
      (on_station ?cuttable_p4 ?station_p2)
      (top_surface_clear ?cuttable_p4 )
      (not (chopped ?cuttable_p4 ))
      (not (holding ?robot_p0 ?cuttable_p4))
      (not (holding ?robot_p0 ?pickupable_p6))
      (not (stacked_on ?pickupable_p3 ?cuttable_p4))
   )
:effect (and
      (chopped ?cuttable_p4 )
 )
)

(:action Cook_1
:parameters ( ?cookable_p5 - cookable ?pickupable_p3 - pickupable ?pickupable_p4 - pickupable
?pickupable_p6 - pickupable ?robot_p0 - robot ?station_p1 - station )
```

```
:precondition (and
      (hand_empty ?robot_p0 )
      (on_station ?cookable_p5 ?station_p1)
      (on_stove_surface ?cookable_p5 )
      (top_surface_clear ?cookable_p5 )
      (not (cooked ?cookable_p5))
      (not (holding ?robot_p0 ?cookable_p5))
      (not (holding ?robot_p0 ?pickupable_p3))
      (not (holding ?robot_p0 ?pickupable_p4))
      (not (holding ?robot_p0 ?pickupable_p6))
      (not (stacked_on ?cookable_p5 ?pickupable_p6))
      (not (stacked_on ?pickupable_p6 ?cookable_p5))
)
:effect (and
      (cooked ?cookable_p5 )
 )
)
```

```
# Learned Predicates & Operators of Franka domain

# Predicates
  name: gripper_empty
  types:
  - robot
  semantic: the robot's gripper is visually not holding any object.

  name: holding
  types:
  - robot
  - pickupable
  semantic: the robot's gripper is visually grasping the specified pickupable object.

  name: mug_full
  types:
  - container
  semantic: the mug container's visible contents reach the inner rim or above (no empty
  interior region is visible above the contents).

  name: plate_top_unoccupied
  types:
  - plate
  semantic: no object is visibly resting in contact with the upper surface of the specified
  plate (its food area is empty).

  name: stacked_on
  types:
  - pickupable
  - plate
  semantic: the specified pickupable object is visually resting in direct contact with the
  upper surface of the specified plate.

  name: plate_is_dirty
  types:
  - plate
  semantic: the specified plate's upper surface contains visible food traces whose appearance
  differs from the plate's base surface (visible residue to be wiped).

# Operators
(:action Pick_1
:parameters ( ?pickupable_p1 - pickupable ?robot_p2 - robot )
:precondition (and
      (gripper_empty ?robot_p2 )
      (not (holding ?robot_p2 ?pickupable_p1))
)
:effect (and
      (holding ?robot_p2 ?pickupable_p1)
      (not (gripper_empty ?robot_p2 ))
 )
)

(:action Pick_2
:parameters ( ?pickupable_p0 - pickupable ?pickupable_p1 - pickupable ?pickupable_p3 -
pickupable ?plate_p4 - plate ?robot_p2 - robot )
:precondition (and
      (gripper_empty ?robot_p2 )
      (stacked_on ?pickupable_p1 ?plate_p4)
      (not (holding ?robot_p2 ?pickupable_p0))
      (not (holding ?robot_p2 ?pickupable_p1))
      (not (holding ?robot_p2 ?pickupable_p3))
      (not (plate_is_dirty ?plate_p4 ))
```

```
      (not (plate_top_unoccupied ?plate_p4 ))
)
:effect (and
      (holding ?robot_p2 ?pickupable_p1)
      (plate_top_unoccupied ?plate_p4 )
      (not (gripper_empty ?robot_p2 ))
      (not (stacked_on ?pickupable_p1 ?plate_p4))
 )
)

(:action Place_1
:parameters ( ?pickupable_p1 - pickupable ?robot_p2 - robot )
:precondition (and
      (holding ?robot_p2 ?pickupable_p1)
      (not (gripper_empty ?robot_p2 ))
)
:effect (and
      (gripper_empty ?robot_p2 )
      (not (holding ?robot_p2 ?pickupable_p1))
 )
)

(:action Pour_1
:parameters ( ?container_p0 - container ?pickupable_p0 - pickupable ?pickupable_p3 - pourable
?robot_p2 - robot )
:precondition (and
      (holding ?robot_p2 ?pickupable_p3)
      (not (gripper_empty ?robot_p2 ))
      (not (holding ?robot_p2 ?pickupable_p0))
      (not (mug_full ?container_p0 ))
)
:effect (and
      (mug_full ?container_p0 )
 )
)

(:action Stack_1
 :parameters (?pickupable_p0 - pickupable
           ?pickupable_p1 - pickupable
           ?pickupable_p3 - pickupable
           ?plate_p4 - plate
           ?robot_p2 - robot)
 :precondition (and
      (holding ?robot_p2 ?pickupable_p1)
      (plate_top_unoccupied ?plate_p4)
      (not (gripper_empty ?robot_p2))
      (not (holding ?robot_p2 ?pickupable_p0))
      (not (holding ?robot_p2 ?pickupable_p3))
      (not (plate_is_dirty ?plate_p4))
      (not (stacked_on ?pickupable_p1 ?plate_p4)))
 :effect (and
      (gripper_empty ?robot_p2)
      (stacked_on ?pickupable_p1 ?plate_p4)
      (not (holding ?robot_p2 ?pickupable_p1))
      (not (plate_top_unoccupied ?plate_p4))))

(:action Wipe_1
:parameters ( ?pickupable_p0 - sponge ?plate_p4 - plate ?robot_p2 - robot )
:precondition (and
      (holding ?robot_p2 ?pickupable_p0)
      (plate_is_dirty ?plate_p4 )
      (not (gripper_empty ?robot_p2 ))
)
:effect (and
      (not (plate_is_dirty ?plate_p4 ))
 )
)
```

```
# Learned Predicates & Operators of Bi-manual Kuka domain

# Predicates
  name: InLeftGripper
  types:
  - robot
  - openable
  semantic: the openable object is currently grasped by the robot's left gripper, with the
  gripper fingers in contact enclosing the object.

  name: InRightGripper
  types:
```

```
  - robot
  - utensil
  semantic: the utensil is physically enclosed and held by the robot's right gripper, with the
  gripper fingers in contact around the utensil.

  name: RightGripperEmpty
  types:
  - robot
  semantic: the robot's right gripper is visibly not holding anything. the fingers are not
  enclosing or contacting any part of the utensil, and the grasping space is empty.

  name: LidOff
  types:
  - openable
  semantic: the openable object's lid/cover is not attached; its opening is exposed and the
  interior is visible.

  name: LeftGripperEmpty
  types:
  - robot
  semantic: the robot's left gripper is visibly not holding any object. the fingers are not
  enclosing or contacting anything and the grasping space is clear.

  name: InContainer
  types:
  - utensil
  semantic: the utensil is located within another object's interior volume, with part of it
  below that object's rim and laterally enclosed by its walls.

  name: OpenableOnTable
  types:
  - openable
  semantic: the openable object's base is in contact with and supported from below by the
  tabletop (resting stably on the surface).

  name: UtensilOnTable
  types:
  - utensil
  semantic: the utensil is resting on and supported by the tabletop, with part of it visibly
  contacting the table surface (i.e., not enclosed within another object).

  name: Closed
  types:
  - openable
  semantic: the openable's lid is visibly attached and covering its opening; the interior is
  not visible.

  name: Coated
  types:
  - utensil
  semantic: a visible layer or clump of material adheres to the utensil's working end (e.g.,
  the blade shows a smear that was absent before).

  name: SpreadOn
  types:
  - food
  semantic: the food's top surface shows a visible layer or patch of spread material (e.g., a
  smear) on it.

  name: HeldByRobot
  types:
  - robot
  - openable
  semantic: the openable object is currently enclosed and held by any of the robot's grippers
  (left or right), with the gripper fingers in contact around the object.

# Operators
(:action LeftArmPick_1
:parameters ( ?openable_p0 - openable ?robot_p1 - robot )
:precondition (and
      (Closed ?openable_p0 )
      (LeftGripperEmpty ?robot_p1 )
      (OpenableOnTable ?openable_p0 )
      (not (HeldByRobot ?robot_p1 ?openable_p0))
      (not (InLeftGripper ?robot_p1 ?openable_p0))
      (not (LidOff ?openable_p0 ))
)
:effect (and
      (HeldByRobot ?robot_p1 ?openable_p0)
      (InLeftGripper ?robot_p1 ?openable_p0)
      (not (LeftGripperEmpty ?robot_p1 ))
```

```
      (not (OpenableOnTable ?openable_p0 ))
 )
)

(:action RightArmPick_1
:parameters ( ?robot_p1 - robot ?utensil_p2 - utensil )
:precondition (and
      (InContainer ?utensil_p2 )
      (RightGripperEmpty ?robot_p1 )
      (not (Coated ?utensil_p2 ))
      (not (InRightGripper ?robot_p1 ?utensil_p2))
      (not (UtensilOnTable ?utensil_p2 ))
)
:effect (and
      (InRightGripper ?robot_p1 ?utensil_p2)
      (not (InContainer ?utensil_p2 ))
      (not (RightGripperEmpty ?robot_p1 ))
 )
)

(:action Drop_1
:parameters ( ?robot_p1 - robot ?utensil_p2 - utensil )
:precondition (and
      (InRightGripper ?robot_p1 ?utensil_p2)
      (not (InContainer ?utensil_p2 ))
      (not (RightGripperEmpty ?robot_p1 ))
      (not (UtensilOnTable ?utensil_p2 ))
)
:effect (and
      (RightGripperEmpty ?robot_p1 )
      (UtensilOnTable ?utensil_p2 )
      (not (InRightGripper ?robot_p1 ?utensil_p2))
 )
)

(:action Open_1
:parameters ( ?openable_p0 - openable ?robot_p1 - robot ?utensil_p2 - utensil )
:precondition (and
      (Closed ?openable_p0 )
      (HeldByRobot ?robot_p1 ?openable_p0)
      (InLeftGripper ?robot_p1 ?openable_p0)
      (RightGripperEmpty ?robot_p1 )
      (not (InRightGripper ?robot_p1 ?utensil_p2))
      (not (LeftGripperEmpty ?robot_p1 ))
      (not (LidOff ?openable_p0 ))
      (not (OpenableOnTable ?openable_p0 ))
)
:effect (and
      (LidOff ?openable_p0 )
      (not (Closed ?openable_p0 ))
 )
)

(:action Scoop_1
 :parameters (?openable_p0 - openable
           ?robot_p1 - robot
           ?utensil_p2 - utensil)
 :precondition (and
      (HeldByRobot ?robot_p1 ?openable_p0)
      (InLeftGripper ?robot_p1 ?openable_p0)
      (InRightGripper ?robot_p1 ?utensil_p2)
      (LidOff ?openable_p0)
      (not (Closed ?openable_p0))
      (not (Coated ?utensil_p2))
      (not (InContainer ?utensil_p2))
      (not (LeftGripperEmpty ?robot_p1))
      (not (OpenableOnTable ?openable_p0))
      (not (RightGripperEmpty ?robot_p1))
      (not (UtensilOnTable ?utensil_p2)))
 :effect (and
      (Coated ?utensil_p2)))

(:action Spread_1
:parameters ( ?food_p0 - food ?openable_p0 - openable ?robot_p1 - robot ?utensil_p2 - utensil
)
:precondition (and
      (Coated ?utensil_p2 )
      (HeldByRobot ?robot_p1 ?openable_p0)
      (InLeftGripper ?robot_p1 ?openable_p0)
      (InRightGripper ?robot_p1 ?utensil_p2)
      (not (InContainer ?utensil_p2 ))
```

```
      (not (LeftGripperEmpty ?robot_p1 ))
      (not (RightGripperEmpty ?robot_p1 ))
      (not (SpreadOn ?food_p0 ))
      (not (UtensilOnTable ?utensil_p2 ))
)
:effect (and
      (SpreadOn ?food_p0 )
 )
)
```

Additionally, we present here predicates and operators written by the PDDL expert for each environment:

```
# Predicates & Operators Written by PDDL Expert


## Robotouille Domain
# Predicates
 name: hand_empty
 types:
 null
 semantic: The agent's hand is empty.

 name: is_holding
 types:
 - pickupable
 semantic: The agent is holding a `pickupable` object.

 name: obj_free
 types:
 - pickupable
 semantic: A `pickupable` object has nothing on top.

 name: station_free
 types:
 - station
 semantic: A `station` is free and has nothing on top.

 name: is_on_top
 types:
 - pickupable
 - pickupable
 semantic: The first `pickupable` object is right on top of the second `pickupable` object
 with nothing in between.

 name: is_on_station
 types:
 - pickupable
 - station
 semantic: A `pickupable` object is on top of a `station`.

 name: is_cut
 types:
 - cuttable
 semantic: The `cuttable` object is cut or sliced.

 name: is_cooked
 types:
 - cookable
 semantic: The `cookable` object is cooked and ready for consumption or further meal
 preparation.



# Operators
(:action Pick
:parameters (?robot - robot ?pickupable - pickupable)
:precondition (and
      (hand_empty)
      (obj_free ?pickupable)
      (not (is_holding ?pickupable))
)
:effect (and
      (not (hand_empty))
      (not (obj_free ?pickupable))
      (is_holding ?pickupable)
 ))

(:action PickFromStack
:parameters (?robot - robot ?top - pickupable ?bot - pickupable)
```

```
:precondition (and
      (hand_empty)
      (not (is_holding ?top))
      (obj_free ?top)
      (is_on_top ?top ?bot)
)
:effect (and
      (not (hand_empty))
      (is_holding ?top)
      (obj_free ?bot)
      (not (obj_free ?top))
      (not (is_on_top ?top ?bot))
)
)

(:action PickFromStation
:parameters (?robot - robot ?top - pickupable ?bot - station)
:precondition (and
      (hand_empty)
      (not (is_holding ?top))
      (obj_free ?top)
      (is_on_station ?top ?bot)
)
:effect (and
      (not (hand_empty))
      (is_holding ?top)
      (station_free ?bot)
      (not (obj_free ?top))
      (not (is_on_station ?top ?bot))
)
)

(:action Place
:parameters (?robot - robot ?pickupable - pickupable ?station - station)
:precondition (and
      (is_holding ?pickupable)
      (station_free ?station)
      (not (obj_free ?pickupable))
)
:effect (and
      (not (is_holding ?pickupable))
      (obj_free ?pickupable)
      (hand_empty)
      (not (station_free ?station))
      (is_on_station ?pickupable ?station)
)
)

(:action Stack
:parameters (?robot - robot ?top - pickupable ?bot - pickupable)
:precondition (and
      (is_holding ?top)
      (obj_free ?bot)
      (not (obj_free ?top))
)
:effect (and
      (not (is_holding ?top))
      (hand_empty)
      (obj_free ?top)
      (not (obj_free ?bot))
      (is_on_top ?top ?bot)
)
)

(:action Cut
:parameters (?robot - robot
      ?cuttable - cuttable
      ?board - cuttingboard)
:precondition (and
      (hand_empty)
      (obj_free ?cuttable)
      (is_on_station ?cuttable ?board)
      (not (is_cut ?cuttable)))
:effect (and
      (is_cut ?cuttable)))

(:action Cook
:parameters (?robot - robot ?cookable - cookable ?cooker - cooker)
:precondition (and
      (hand_empty)
      (obj_free ?cookable)
```

```
      (is_on_station ?cookable ?cooker)
      (not (is_cooked ?cookable))
)
:effect (and
      (is_cooked ?cookable)
)
)


## Franka Domain
# Predicates
  name: hand_empty
  types:
  - robot
  semantic: The `robot` is not holding anything in its gripper.

  name: is_holding
  types:
  - robot
  - pickupable
  semantic: The `robot` is holding a `pickupable` object in its gripper.

  name: is_clean
  types:
  - plate
  semantic: The `plate` object is cleaned after wiping.

  name: is_on_top
  types:
  - pickupable
  - plate
  semantic: A `pickupable` object is on top of a `plate` object.

  name: plate_empty
  types:
  - plate
  semantic: A `plate` object has nothing on top of it.

  name: container_filled
  types:
  - container
  semantic: A `container` has something inside of it.

# Operators
(:action Pick
:parameters (?robot - robot ?pickupable - pickupable)
:precondition (and
      (hand_empty ?robot)
      (not (is_holding ?robot ?pickupable))
)
:effect (and
      (not (hand_empty ?robot))
      (is_holding ?robot ?pickupable)
)
)

(:action PickFromTable
:parameters (?robot - robot ?pickupable - pickupable ?plate - plate)
:precondition (and
      (hand_empty ?robot)
      (not (is_holding ?robot ?pickupable))
      (is_on_top ?pickupable ?plate)
      (not (plate_empty ?plate))
)
:effect (and
      (not (hand_empty ?robot))
      (is_holding ?robot ?pickupable)
      (not (is_on_top ?pickupable ?plate))
      (plate_empty ?plate)
)
)

(:action Place
:parameters (?robot - robot ?pickupable - pickupable)
:precondition (and
      (not (hand_empty ?robot))
      (is_holding ?robot ?pickupable)
)
:effect (and
      (hand_empty ?robot)
      (not (is_holding ?robot ?pickupable))
```

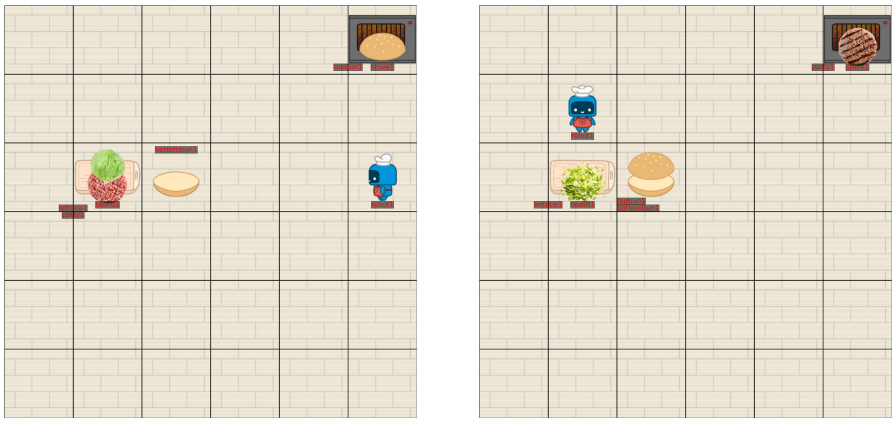
```
)
)

(:action Pour
:parameters (?robot - robot ?pourable - pourable ?container - container)
:precondition (and
      (is_holding ?robot ?pourable)
      (not (container_filled ?container))
)
:effect (and
      (container_filled ?container)
)
)

(:action Stack
:parameters (?robot - robot ?pickupable - pickupable ?plate - plate)
:precondition (and
      (not (hand_empty ?robot))
      (is_holding ?robot ?pickupable)
      (not (is_on_top ?pickupable ?plate))
      (plate_empty ?plate)
      (is_clean ?plate)
)
:effect (and
      (hand_empty ?robot)
      (not (is_holding ?robot ?pickupable))
      (is_on_top ?pickupable ?plate)
      (not (plate_empty ?plate))
)
)

(:action Wipe
:parameters (?robot - robot ?sponge - sponge ?plate - plate)
:precondition (and
      (is_holding ?robot ?sponge)
      (not (is_clean ?plate))
      (plate_empty ?plate)
)
:effect (and
      (is_clean ?plate)
)
)

## Bi-manual Kuka Domain
# Predicates
 name: lefthand_empty
 types:
 - robot
 semantic: The `robot` is not holding anything in its left hand.

 name: righthand_empty
 types:
 - robot
 semantic: The `robot` is not holding anything in its right hand.

 name: in_lefthand
 types:
 - robot
 - openable
 semantic: The `robot` is holding an `openable` object in its left hand.

 name: in_righthand
 types:
 - robot
 - utensil
 semantic: The `robot` is holding an `utensil` object in its right hand.

 name: is_opened
 types: !!python/tuple
 - openable
 semantic: An `openable` object (such as a jar) is opened.

 name: is_clean
 types:
 - utensil
 semantic: A `utensil` object (such as a knife) is clean.

 name: is_upright
 types:
 - utensil
 semantic: A `utensil` object (such as a knife) is upright and graspable.
```

```
  name: contains_ingredient
  types: !!python/tuple
  - utensil
  semantic: A `utensil` object contains a spreadable ingredient.

  name: spread_on_top
  types:
  - food
  semantic: The `food` object contains a spreadable ingredient on top of it.

# Operators
(:action LeftArmPick
:parameters (?robot - robot ?openable - openable)
:precondition (and
      (lefthand_empty ?robot)
      (not (in_lefthand ?robot ?openable))
)
:effect (and
      (not (lefthand_empty ?robot))
      (in_lefthand ?robot ?openable)
)
)

(:action RightArmPick
:parameters (?robot - robot ?utensil - utensil)
:precondition (and
      (righthand_empty ?robot)
      (is_upright ?utensil)
      (not (in_righthand ?robot ?utensil))
)
:effect (and
      (not (righthand_empty ?robot))
      (not (is_upright ?utensil))
      (in_righthand ?robot ?utensil)
)
)

(:action Drop
:parameters (?robot - robot ?utensil - utensil)
:precondition (and
      (not (righthand_empty ?robot))
      (in_righthand ?robot ?utensil)
)
:effect (and
      (righthand_empty ?robot)
      (not (in_righthand ?robot ?utensil))
)
)

(:action Open
:parameters (?robot - robot ?openable - openable)
:precondition (and
      (not (is_opened ?openable))
      (in_lefthand ?robot ?openable)
      (not (lefthand_empty ?robot))
      (righthand_empty ?robot)
)
:effect (and
      (is_opened ?openable)
)
)

(:action Scoop
:parameters (?robot - robot ?knife - utensil ?jar - openable)
:precondition (and
      (in_lefthand ?robot ?jar)
      (in_righthand ?robot ?knife)
      (is_opened ?jar)
      (is_clean ?knife)
      (not (contains_ingredient ?knife))
)
:effect (and
      (contains_ingredient ?knife)
      (not (is_clean ?knife))
)
)

(:action Spread
:parameters (?robot - robot ?knife - utensil ?bread - food)
:precondition (and
```

```
      (in_righthand ?robot ?knife)
      (contains_ingredient ?knife)
      (not (spread_on_top ?bread))
)
:effect (and
      (not (contains_ingredient ?knife))
      (spread_on_top ?bread)
)
)
```

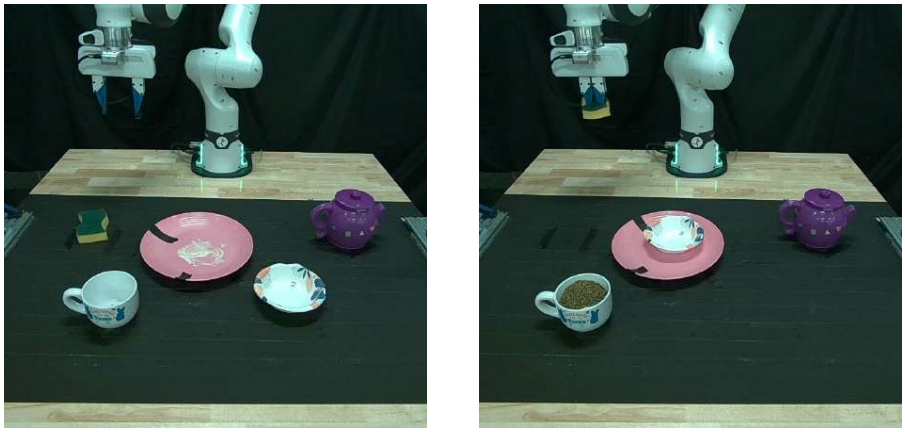
G.3 Example Tasks

We provide examples of the skill planning problems for evaluation. Specifically, Robotouille uses images generated by the simulator, Franka uses images taken by a camera in front of the robot, and Bimanual Kuka uses images taken by its egocentric camera. For impossible tasks, the impossibilities can be caused by not having a intermediate place for stack permutation (Fig. 9a) or limiting by the current skill set (Fig. 9b), e.g., cleaning up a filled mug or spreading ranch on the plate.



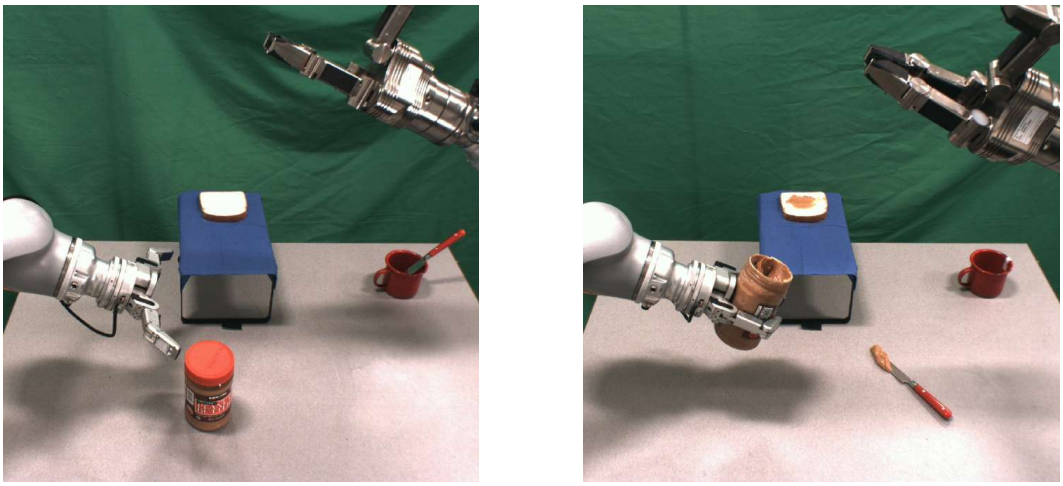
(a) Initial state (b) Goal state

Figure 6: Example task in Robotouille.



(a) Initial state (b) Goal state

Figure 7: Example task in Franka.

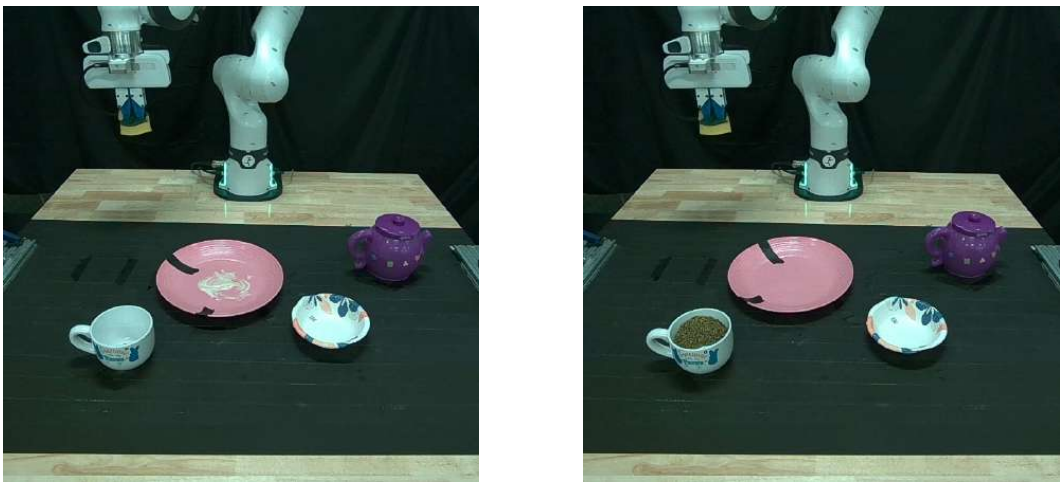


(a) Initial state (b) Goal state

Figure 8: Example task in Bimanual Kuka.



(a) Initial state (b) Goal state



(c) Initial state (d) Goal state

Figure 9: Example Impossible Tasks In Robotouille.

H VLM Reliability Study

H.1 Classification Accuracy

We here analyze the classification accuracy of the vision-language model (VLM) used in both robotic experiments. Since the predicates are generated on the fly by the VLM, we do not have the ground truth values for them, and thus we must verify if the truth values match the images manually.

Per-predicate Classification Evaluation Since predicates are originally lifted and can be grounded with different combinations of objects, we first define a classification over a low-level state of a grounded predicate as correct if (1) all parameters appear in the scene (if the predicate is not nullary) and (2) the truth value of the predicate match the low-level state specified by the image input. Then, we define a classification of a lifted predicate over a low-level state as correct if all of its grounded instances are classified correctly over that state.

Results and Analysis Over all predicates, the classification accuracy is 86.7% for the Franka experiment, and 98.5% for the bimanual Kuka experiment. Compared to the planning performance reported for both experiments, the classification accuracy is generally much higher. One reason for this mismatch is that, due to the rigidity of symbolic planning, even flipping the truth value of a single predicate can cause planning failure. To investigate further, we identify predicates with poor classification accuracy and analyze how they harm planning performance in the next paragraph.

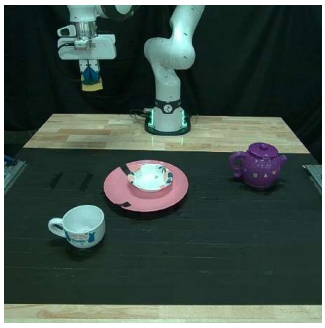
Per-predicate Accuracy The learned symbolic model of the Franka experiment contains 6 predicates, which have 11 possible grounded instances. The learned symbolic model of the bimanual Kuka experiment contains 12 predicates, which have 13 possible grounded instances. We evaluate per-predicate accuracy for both in Table 4 and Table 5. From the results of the Franka experiment, we identify the two predicates, `gripper_empty` and `holding`, that caused all planning failures, and they fail almost simultaneously due to their semantic correlation. With further investigation, we found that the misclassifications were induced by a single object, `Sponge`, which is possibly due to the color of the object and the background being too similar. In the bimanual Kuka experiment, it is `coated` (if the knife has peanut butter on it) that caused most of the planning failure, likely caused by the lighting conditions. These observations suggest the accuracy of VLM is a limiting factor, and resolving them poses a promising path to improving the performance of SkillWrapper.

Table 4: Per-predicate accuracy of Franka.

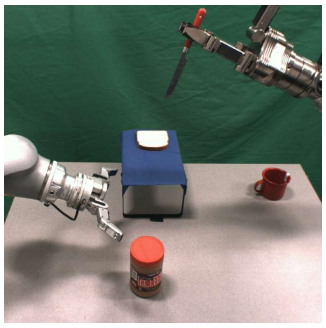
	gripper_empty	holding	mug_full	plate_top_unoccupied	stacked_on	plate_is_dirty
Accuracy (%)	60.0	60.0	100.0	100.0	100.0	100.0

Table 5: Per-predicate accuracy of Bimanual Kuka.

	InLeftGripper	InRightGripper	RightGripperEmpty	LeftGripperEmpty	LidOff	InContainer
Accuracy (%)	100.0	100.0	100.0	100.0	100.0	98.3
	OpenableOnTable	Closed	Coated	SpreadOn	HeldByRobot	UtensilOnTable
Accuracy (%)	96.7	100.0	88.3	98.3	100.0	100.0



(a) ✗`holding(Sponge)` = **F**



(b) ✗`coated(Knife)` = **T**

H.2 Real-world Robustness

To evaluate the real-world robustness of the VLM, we additionally conduct experiments to investigate factors such as viewpoints, lighting conditions, or domain shifts. For each of them, we collect a held-out set of images by varying these factors. We report per-predicate accuracy, and all numbers are averaged across three individual runs.

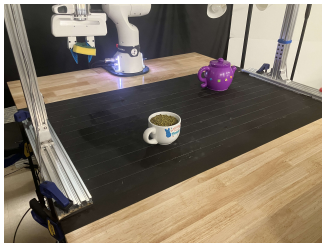
Viewpoints We collect visual observation data from two viewpoints and sample five configurations for each viewpoint. From the results of Franka experiments, we observe that the classification accuracies of certain predicates are higher from the viewpoint closer to the corresponding objects: at viewpoint #1, all predicates can be perfectly classified, while predicates involving gripper or mug, such as `gripper_empty`, `holding` and `mug_full`, are significantly lower from viewpoint #2, which is farther from the objects. In bimanual Kuka experiments, the result is mostly stable across different viewpoints and generally better than in the Franka environment, which is possibly due to fewer background distractions. Though the accuracy varies across viewpoints, its performance remains reliable as long as the full observability assumption still holds.

Table 6: Per-predicate accuracy of Franka at Viewpoint #1.

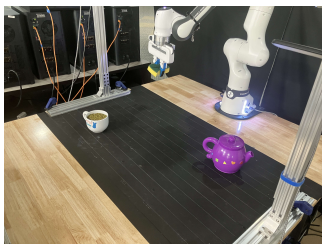
	gripper_empty	holding	mug_full	plate_top_unoccupied	stacked_on	plate_is_dirty
Accuracy (%)	100.0	100.0	100.0	100.0	100.0	100.0

Table 7: Per-predicate accuracy of Franka at Viewpoint #2.

	gripper_empty	holding	mug_full	plate_top_unoccupied	stacked_on	plate_is_dirty
Accuracy (%)	80.0	80.0	90.0	100.0	100.0	100.0



(a) ✓`holding(Sponge)` = **T**



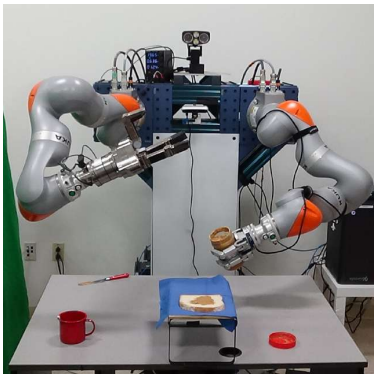
(b) ✗`holding(Sponge)` = **F**

Table 8: Per-predicate accuracy of Bimanual Kuka at Viewpoint #1.

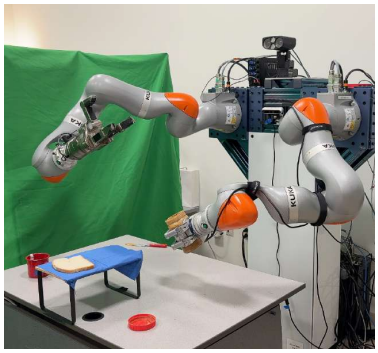
	InLeftGripper	InRightGripper	RightGripperEmpty	LeftGripperEmpty	LidOff	InContainer
Accuracy (%)	100.0	100.0	100.0	100.0	100.0	100.0
	OpenableOnTable	Closed	Coated	SpreadOn	HeldByRobot	UtensilOnTable
Accuracy (%)	100.0	100.0	100.0	100.0	100.0	100.0

Table 9: Per-predicate accuracy of Bimanual Kuka at Viewpoint #2.

	InLeftGripper	InRightGripper	RightGripperEmpty	LeftGripperEmpty	LidOff	InContainer
Accuracy (%)	100.0	100.0	100.0	100.0	100.0	100.0
	OpenableOnTable	Closed	Coated	SpreadOn	HeldByRobot	UtensilOnTable
Accuracy (%)	100.0	100.0	100.0	100.0	100.0	93.3



(a) ✓ `UtensilOnTable(Knife)` = **T**



(b) ✗ `UtensilOnTable(Knife)` = **F**

Lighting conditions We collect visual observation data under two lighting conditions and sample five configurations for each one. We find that the VLM is generally robust to different lighting conditions, except for several extremely difficult ones, such as under lighting condition #2 in Bimanual Kuka, where the objects are heavily shadowed.

Table 10: Per-predicate accuracy of Franka under Lighting Condition #1.

	gripper_empty	holding	mug_full	plate_top_unoccupied	stacked_on	plate_is_dirty
Accuracy (%)	90.0	90.0	100.0	96.7	100.0	100.0

Table 11: Per-predicate accuracy of Franka under Lighting Condition #2.

	gripper_empty	holding	mug_full	plate_top_unoccupied	stacked_on	plate_is_dirty
Accuracy (%)	90.0	90.0	100.0	98.3	100.0	100.0



(a) ✓ `stacked_on(Teapot, Plate)` = **T**



(b) ✓ `stacked_on(Teapot, Plate)` = **T**

Table 12: Per-predicate accuracy of Bimanual Kuka under Lighting Condition #1.

	InLeftGripper	InRightGripper	RightGripperEmpty	LeftGripperEmpty	LidOff	InContainer
Accuracy (%)	100.0	100.0	100.0	100.0	100.0	100.0
	OpenableOnTable	Closed	Coated	SpreadOn	HeldByRobot	UtensilOnTable
Accuracy (%)	100.0	100.0	100.0	100.0	100.0	100.0

Table 13: Per-predicate accuracy of Bimanual Kuka under Lighting Condition #2.

	InLeftGripper	InRightGripper	RightGripperEmpty	LeftGripperEmpty	LidOff	InContainer
Accuracy (%)	100.0	80.0	80.0	100.0	100.0	73.3
	OpenableOnTable	Closed	Coated	SpreadOn	HeldByRobot	UtensilOnTable
Accuracy (%)	100.0	100.0	100.0	86.7	100.0	100.0



(a) ✓ `RightGripperEmpty()` = **T**

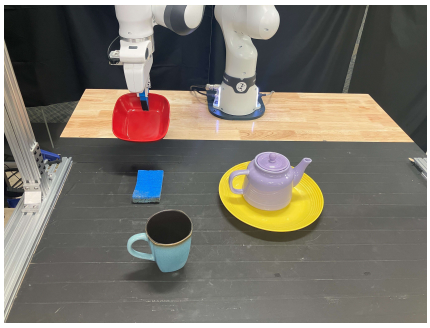


(b) ✗ `RightGripperEmpty()` = **F**

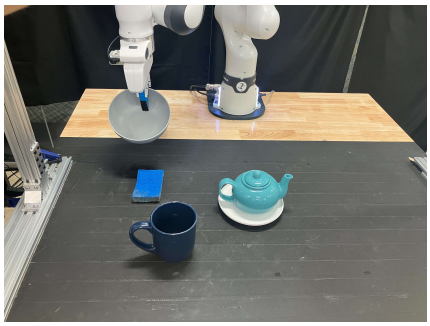
Domain shift SkillWrapper relies entirely on semantics to prompt the VLM, abstracting raw states into symbolic states without using visual features. The only requirement for generalizing to novel objects is that the VLM can correctly identify the object referents in the image based on their type information provided in the language prompt. To evaluate this generalization capability, we collected visual observation data (five images per environment) under domain shift by swapping objects with new instances and sampling two configurations. From this observation, we found that the only failure mode introduced by domain shifts occurs when the VLM cannot recognize an object because its visual appearance does not align with the semantics. For example, a plate that is too small might be misclassified as a saucer, leading to incorrect symbolic states.

Table 14: Per-predicate accuracy of Franka under Domain Shift.

	gripper_empty	holding	mug_full	plate_top_unoccupied	stacked_on	plate_is_dirty
Accuracy (%)	90.0	90.0	100.0	100.0	73.3	100.0



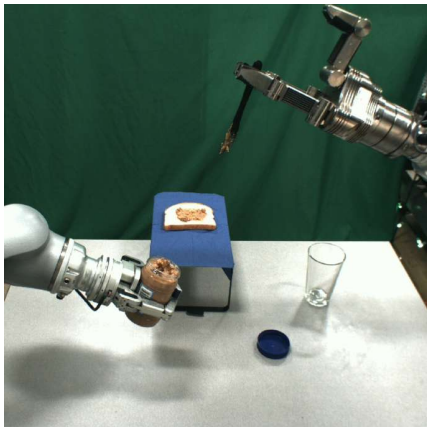
(a) ✓ `stacked_on(Teapot, Plate)` = **T**



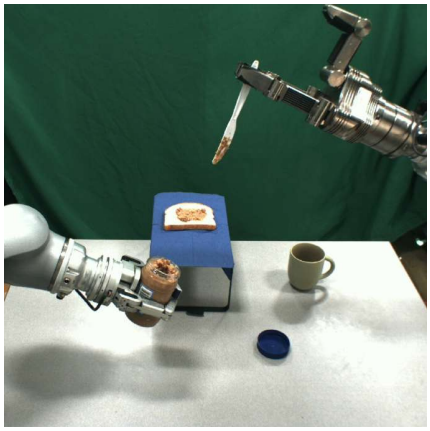
(b) ✗ `stacked_on(Teapot, Plate)` = **F**

Table 15: Per-predicate accuracy of Bimanual Kuka under Domain Shift.

	InLeftGripper	InRightGripper	RightGripperEmpty	LeftGripperEmpty	LidOff	InContainer
Accuracy (%)	100.0	100.0	100.0	100.0	100.0	100.0
	OpenableOnTable	Closed	Coated	SpreadOn	HeldByRobot	UtensilOnTable
Accuracy (%)	100.0	100.0	100.0	100.0	100.0	100.0



(a) ✓ `coated(Knife)` = **T**



(b) ✓ `coated(Knife)` = **T**

H.3 Other VLMs

We further examine the possibility of using open-source VLMs as alternatives for SkillWrapper. We choose Qwen3-VL-235B (Bai et al., 2025) for comparison. To evaluate its capability, we conduct two sets of preliminary experiments: predicate classification and predicate invention.

Predicate Classification We collected a subset of images (five from Franka and ten from Bimanual Kuka) and evaluated the truth values of each predicate with the two models. From the result, we observed that the two models have different failure patterns, and a prominent one is that Qwen3 can reliably detect if the gripper is holding an object, except for occasional classification errors on the objects being held. In general, we found two models perform on par with each other, and thus we believe they can be used interchangeably for predicate classification.

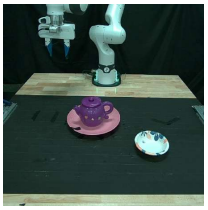
Table 16: Per-predicate accuracy of Franka.

	gripper_empty	holding	mug_full	plate_top_unoccupied	stacked_on	plate_is_dirty
GPT-5 Acc. (%)	60.0	60.0	100.0	100.0	100.0	100.0
Qwen3 Acc. (%)	100.0	80.0	100.0	100.0	100.0	80.0

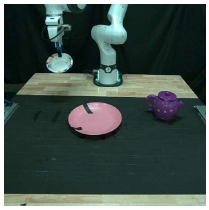
Table 17: Per-predicate accuracy of Bimanual Kuka.

	InLeftGripper	InRightGripper	RightGripperEmpty	LeftGripperEmpty	LidOff	InContainer
GPT-5 Acc. (%)	100.0	100.0	100.0	100.0	100.0	100.0
Qwen3 Acc. (%)	100.0	100.0	100.0	100.0	100.0	100.0
	OpenableOnTable	Closed	Coated	SpreadOn	HeldByRobot	UtensilOnTable
GPT-5 Acc. (%)	100.0	100.0	100.0	100.0	100.0	100.0
Qwen3 Acc. (%)	100.0	100.0	90.0	80.0	100.0	100.0

Predicate Invention We qualitatively compare the performance of both models on inventing predicates by reasoning over contrastive pairs of transitions. For each environment, we curated two contrastive pairs, and each model is prompted by the same input to invent one new predicate. A predicate is considered correct if it is a semantic synonym or antonym of the target predicate. From the result, we can conclude that GPT-5 is much more reliable in reasoning over the transitions for predicate invention, and thus Qwen3 cannot be used as an alternative for this specific task. (We omitted `?robot` from all predicates' arguments for simplicity.)



(a) ✓ `PickUp(Bowl)`

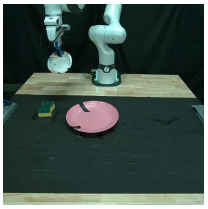


(b) ✗ `PickUp(Teapot)`

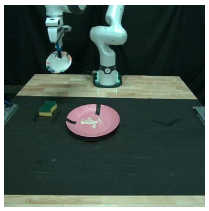
GPT-5
✓ `gripper_is_empty()`
✓ `gripper_is_empty()`
✓ `gripper_is_free()`

Qwen3
✓ `gripper_free()`
✓ `gripper_free()`
✗ `is_visible(?pickupable)`

Figure 17: **Predicate Invention Case #1 in Franka.** *Target predicate*: `GripperEmpty()`
Existing predicates: $\emptyset$



(a) ✓ `Stack(Bowl, Plate)`

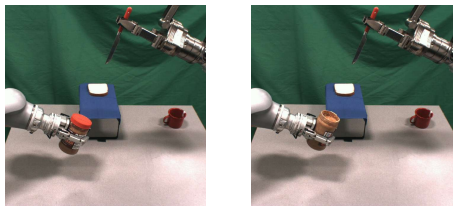


(b) ✗ `Stack(Bowl, Plate)`

GPT-5
✓ `plate_top_empty(?plate)`
✓ `plate_is_clean(?plate)`
✓ `plate_is_clean(?plate)`

Qwen3
✗ `stacked_on (?pickupable, ?plate)`
✗ `on_center_of (?pickupable, ?plate)`
✗ `is_fully_supported (?pickupable, ?plate)`

Figure 18: **Predicate Invention Case #2 in Franka.** *Target predicate*: `PlateIsDirty(?plate)`
Existing predicates: `GripperEmpty()`, `Holding(?pickupable)`



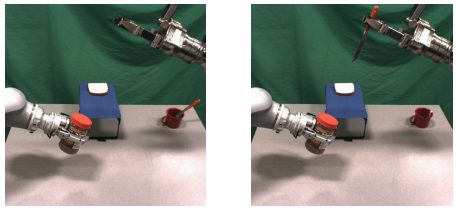
(a) ✓ Scoop(Knife, Jar)

(b) ✗ Scoop(Knife, Jar)

GPT-5
✓ Open(?openable)
✓ Open(?openable)
✓ Open(?openable)

Qwen3
✗ UtensilInOpenable (?utensil, ?openable)
✗ UtensilInOpening (?utensil, ?openable)
✗ UtensilInOpenable (?utensil, ?openable)

Figure 19: **Predicate Invention Case #1 in Bi-Kuka.** *Target predicate*: LidOff(?openable) *Existing predicates*:InLeftGripper(?openable), InRightGripper(?utensil)



(a) ✓ Open(Jar)

(b) ✗ Open(Jar)

GPT-5
✓ RightHandEmpty()
✓ RightHandEmpty()
✗ LidAttached(?openable)

Qwen3
✗ FullyEnclosedByLeftGripper (?openable)
✗ FullyEnclosedByLeftGripper (?openable)
✗ FullyEnclosedByLeftGripper (?openable)

Figure 20: **Predicate Invention Case #2 in Bi-Kuka.** *Target predicate*: RightGripperEmpty() *Existing predicates*:InLeftGripper(?openable), LidOff(?openable)

I Implementation Details

I.1 Planner and Planning Time

We use K* planner (Katz & Lee, 2023) to generate top K optimal plans, where K in practice is the maximum planning budget. We use an i9-13900F CPU for running all the planning tasks. On average, each planning problem takes 0.0599 seconds. Specifically, in Robotouille experiments, easy problems take 0.0549 seconds, hard problems takes 0.0583 seconds, and impossible problems take 0.0565 seconds per problem; in Franka experiments, in-domain problems take 0.0529 seconds, generalization problems take 0.5175 seconds, and impossible tasks take 0.0516 seconds per problem; in Bimanual Kuka experiments, all problems take 0.0553 seconds on average.

I.2 API Call

For running the experiments, we made approximately 9,300 calls to GPT-5, which cost $96.59 in total.

I.3 Hyperparameters

We here report and summarize all hyperparameters of SkillWrapper used for the experiment to provide better reproducibility. For all experiments, we set the batch size of skill sequence proposal to be 5 and interaction budget per iteration to be 15, and we run SkillWrapper for 5 iterations for each individual run, except in the Franka generalization experiment, where we use one iteration in each of the three training environments.. For Robotouille experiments, we set the threshold h to be 0.6. For Franka and bimanual Kuka experiments, we set the threshold h to be 0.5.

I.4 Experiments

Simulated Robotouille We conduct experiments in Robotouille (Gonzalez-Pumariega et al., 2025), a simulated kitchen environment with six objects and a robot with five skills:*Pick*, *Place*, *Cut*, *Cook*, and *Stack*. In the environment, there are several objects: a patty, lettuce, a top bun, and a bottom bun; there is also a cutting board and a stove for cutting the lettuce and cooking the patty. We design and categorize 50 skill planning problems for this domain: 20 easy problems, whose optimal solutions have at most 7 steps; 20 hard problems, whose optimal solutions have at most 15 steps; and 10 impossible tasks that cannot be realized in the environment, which a sound planning model must identify.

Single-Arm Manipulation We employ a Franka Emika Research 3 robotic arm equipped with a UMI gripper (Chi et al., 2024). The workspace is observed by a single Intel RealSense D455 exocentric RGB-D camera, oriented to capture both the tabletop scene and the robot. The RGB data from this camera are used for learning symbolic models, while the depth information supports object pose estimation. Object poses are estimated using FoundationPose (Wen et al., 2024), which leverages high-fidelity 3D scanned models of the target objects. System-level communication and coordination are implemented in ROS 2 (Humble), which interfaces with motion planning, perception, and control modules. This setup supports five parameterized skills: *Pick*, *Place*, *Stack*, *Pour*, and *Wipe*. The first four skills (*Pick*, *Place*, *Stack*, and *Pour*) are executed through motion planning with the MoveIt framework, conditioned on both the end-effector and object poses. The *Wipe* skill is implemented by replaying a teleoperated trajectory. The object set $\mathcal{O}$ contains five objects: a mug, a teapot, a plate, a bowl, and a sponge. The robot can pick up and place all objects except the mug and plate, pour ingredients from the teapot into the mug, and use the sponge to wipe the plate if it is dirty.

Bimanual Manipulation We use a robot with two horizontally mounted KUKA LBR iiwa 7 R800 manipulators, one with a BarrettHand BH8-282 gripper, and the other with a Schunk Dextrous Hand 2.0 gripper. The robot collects RGB data used for learning symbolic models with a MultiSense S7 camera mounted on a Pan-Tilt unit, while using an Intel RealSense D455 camera for RGB-D data used in pose estimation (Wen et al., 2024) of the objects in the scene. We use ROS 1 and KUKA FRI to communicate with the robot and utilize the built-in joint impedance control with position target as the low-level controller. At the high level, we create collision models of all objects in the scene and use a task and motion planner to generate motion plans for each skill. In this setting, the robot is equipped with six skills: *LeftArmPick*, *RightArmPick*, *Open*, *Scoop*, *Spread*, and *Drop*. The

Pick skills (compatible with knife and peanut butter jar) are implemented using motion planning. The *OpenJar*, *Scoop*, and *Spread* skills are implemented using a combination of motion planning and pre-defined trajectory playback. The object set $\mathcal{O}$ consists of three objects: a peanut butter jar, a knife, and a slice of bread. The robot can pick up the knife and jar from their initial positions, drop the knife, open the jar, scoop peanut butter with the knife, and spread it on the bread. Notably, this environment contains multiple dead ends, which would hinder the data collection process. For example, the knife cannot be picked up again once dropped on tabletop, the jar cannot be released once picked up, and the bread and knife cannot be cleaned once in contact with peanut butter.

I.5 Prompts

In this section, we provide the prompts used for the core components of SkillWrapper (specifically skill sequence proposal, predicate invention, and predicate evaluation) as well as the ViLA (Hu et al., 2023) baseline. For predicate evaluation (Appendix I.5), we empirically observed that it is more accurate when evaluation is done in batches, where the truth values of multiple predicates are evaluated at once rather than one at a time. In addition, when asking for a fixed and structured output, the accuracy is significantly lower than a free-form output. Therefore, we adopt a two-stage evaluation process: in the first stage, the foundation model generates a response in any format, and in the second stage, it provides a summary of the output from the previous step.

Skill Sequence Proposal

System Prompt

```
<AGENT_DESCRIPTION> is attempting to learn the preconditions and
effects for a finite set of skills by executing exploratory skill
sequences and exploring the environment.
```

Skill Sequence Proposal Prompt

```
Propose a set of skill sequences for a robot to execute.  The robot
is attempting to learn the preconditions and effects for a finite
set of operators.  The robot can navigate the environment freely
but only has one gripper.  The robot has access to the following
skills with their associated arguments:

[SKILL_PROMPT]

The list of objects the robot has previously encountered in the
environment are:

[OBJECT_IN_SCENE]
[ENV_DESCRIPTION]

The pairs of consecutive skills (skill1, skill2) that have been
least explored are:  [[LEAST_EXPLORED_SKILLS]]. Certain skills
have similar names and arguments, but different preconditions and
effects.  Using the list of objects and the skill preconditions /
effects learned, generate 5 skill sequences and their sequence of
skills such that:
(1) the skill sequences should violate their preconditions
occasionally.
(2) at least 1 unexplored skill pair is used in each skill
sequence.
(3) all skill sequences have at least 15 skills in sequence.
(4) there are no same skills with same arguments consecutively in
the sequence.

Output only the sequence of skills to execute, ensuring to follow
the naming/syntax/arguments for skills provided.  Output 1 skill
every new line, following the format below:

Skill Sequence 1:
GoTo(CounterTop)
PickUp(Apple, CounterTop)

Skill Sequence 2:
```

Predicate Invention

Predicate Invention Prompt

```
[AGENT_DESCRIPTION]

The robot has been programmed with the skill [LIFTED_SKILL]
two times.  In the first execution, the grounded skill
[GROUNDED_SKILL_1] [SUCCESS_1], and in the second execution,
[GROUNDED_SKILL_2] [SUCCESS_2].  The difference in outcomes suggests
that the existing predicate set is insufficient to fully capture
the preconditions for successful execution of this skill.

Your task is to propose a single new high-level predicate and its
semantic meaning based on the visual comparison of the two input
images taken before each execution.

Predicates should meet these criteria:
- The predicate must be grounded in visual state only (e.g.,
"gripper is open," "object is above table," "arm is holding
object").
- Describe object state or spatial relations relevant to task
success (e.g., gripper open/closed, object on left/right of
gripper, object touching/supporting another object, etc.)
- Do not infer properties like affordances (is_graspable), alignment
with grippers, or success likelihood that are vaguely defined and
cannot be clearly determined visually.
- Avoid using concept like grasping zone or robot's reachability to
define the predicate since they are not defined by common sense.
- Use at most 2 parameters (e.g., predicate(x), predicate(x,
y), predicate()), where robot arm must be included for any
robot-environment relation.
- Avoid predicates that assume internal properties like
is_graspable, is_properly_aligned, or any accessibility/reachability
reasoning that cannot be determined visually.
- The semantic meaning should be a grounded and objective
description of the predicate in terms of the physical scene (e.g.,
"the object is fully enclosed by the robot's gripper"), not about
execution success or skill dynamics.
- The parameters of the predicate must be subset of the parameters
of the skill.

Format your output as follows:
`predicate_name(parameters)`:  semantic_meaning.

for example:
`CloseTo(arm, location)`:  the robot arm is close to the location.

Current predicates:  [PRED_LIST]

Previously proposed but rejected predicates:  [TRIED_PRED]

Avoid duplicates or near-duplicates of existing predicates and
rejected predicates.  Reason over using a paragraph and generate
the predicate and the semantic meaning in the given format in a
separate line.

One new predicate candidate for improving the representation of the
precondition for [LIFTED_SKILL] ( Don't use any parameter other than
[PARAMETERS]):
```

Predicate Evaluation

Predicate Evaluation Prompt: Step 1

```
Given the current observation of the simulated kitchen domain,
the object types, and the list of predicates, what are the true
grounded predicates?

[ENVIRONMENTAL_DESCRIPTION]

Objects:
[OBJECTS]

Predicates:

[PREDICATES]
```

Predicate Evaluation Prompt: Step 2

```
Summarize what the true grounded predicates are from this response,
and list them in the format of predicate_name(arg1, arg2, ...)  in
separate lines with no any formatting.  If the response contains
typos of object names or redundant indices, you should correct
them.  Correct object names are:  [OBJECT_NAMES]. If the response
include redundant predicates that are not in this list, you should
filter them.  Correct predicates are:  [PRED_NAMES]. The response
is:

"""
[RESPONSE]
""
```

ViLa

ViLA Prompt

```
You are [AGENT_DESCRIPTION]. As a robot, you are able to execute the
following skills:
[SKILLS]

Here are the objects and their types that are compatible with your
skills:
[OBJECTS]

You are given two images:  The first one captures your current
observation, and the second one specifies your goal.  Given both
images, your job is to generate a plan starting from the *current
state* to the goal state.  You should first reason about the goal
of the task and how the skills can be chained to solve it in the
first paragraph.  After the reasoning, return the plan from the
current state in a new paragraph by listing skills in separate
lines with no additional explanation, header, or numbering.
Use "Done" in the skill list to indicate the task is complete,
and report if the task is impossible to solve by simply returning
"Impossible".
```

J Related Works

SkillWrapper draws ideas from different fields of research, such as model learning, abstraction learning, and task and motion planning (TAMP). Other methods have used FMs as high-level planners (Hu et al., 2023). Concurrent work has also explored how representations can be learned directly from pixels (Athalye et al., 2026). Multiple approaches (Han et al., 2024; Liang et al., 2025) have used foundation models to learn symbolic representations of robot skills, but require extensive feedback or prior knowledge from human experts. To the best of our knowledge, our work is the first to use formally characterized planning model properties to guide generative predicate invention.

TAMP has long been used to solve complex robot tasks (Shah et al., 2020; Garrett et al., 2021). However, these approaches require symbolic models of robot skills for task planning. Various approaches have been developed to learn such symbolic models compatible with TAMP solvers from high-dimensional inputs (Konidaris et al., 2018; Silver et al., 2023; Shah et al., 2025). Additionally, the abstract representations learned through these methods are not directly human-interpretable. In contrast, we explicitly design our approach to work with high-dimensional inputs and generate human-interpretable abstractions using pre-trained FMs.

Abstraction Learning for Robotic Skills There has been a long track of works focusing on building hierarchies that abstract away high-dimensional details with low-dimensional abstractions for planning (Konidaris & Barto, 2009; Konidaris et al., 2018; Shah et al., 2025), and those applied to robotics are usually connected to TAMP (Shah et al., 2020; Garrett et al., 2021). These approaches, however, are incapable of handling high-dimensional sensory-motor signals (such as images) as input. Research on action model learning (Xi et al., 2024; Juba et al., 2021) learn symbolic action models for input skills. However, unlike our method, these approaches require symbols to be provided as input. Similar to our system's integration of self-play and focus on uncovering skill conditions, Verma et al. (2022) focus on assessing capabilities of black-box agents for grid world-like tasks while assuming that the agent is an oracle. A tangential research effort on chaining various skills in novel environments involves training extra models (Yokoyama et al., 2024) and STRIPS task planner with action primitives (Gu et al., 2022; Szot et al., 2021).

Predicate Learning for Robotic Tasks Predicates provide a convenient way to abstract away low-level details of the environment and build efficient and compact representations. Prior to foundation models, previous attempts to build classifiers for predicates from raw image inputs originated from the neuro-symbolic domain (Johnson et al., 2017; Mao et al., 2019), and their initial application for robotics took a similar supervised learning approach with labeled demonstrations (Migimatsu & Bohg, 2022) or generated tasks (Lamanna et al., 2023). After the emergence of foundation models, recent works guide skill learning with predicates generated by LLM or together with human interaction. Li et al. (2024) invents symbolic skills for reward functions used for RL training but cannot generalize to skills learned through latent objectives, which is more commonly seen in imitation learning. Li & Silver (2023) and Han et al. (2024) leverage human experts to provide feedback to the LLM to help it improve the learned predicates and skills.

Task Generation for Robotics The approach of automatically proposing tasks has been studied for active learning and curriculum learning in grid worlds and games (Wang et al., 2019; Jiang et al., 2021) to robotic domains (Fang et al., 2021; 2022). Lamanna et al. (2023) generates tasks in PDDL as training sets to learn classifiers for object properties in predicates format, while they assume the action operators are given. With the commonsense reasoning ability of foundation models, recent works have applied the idea of automatic task proposing and self-playing for exploration (Nasiriany et al., 2024; Ren et al., 2024), data collection (Wang et al., 2024b; Yang et al., 2024; Ahn et al., 2024), boosting skills learning (Ha et al., 2023; Wang et al., 2024a), and scene understanding (Jiang et al., 2025). These works indicate a promising direction for generating robotic data and scaling up. Following the idea, we equipped our system with a task-proposing module for generating skill sequences specific to skills and predicates, which serves the idea of both data collection and exploration.

Embodied Reasoning with Foundation Models There has been a track of work on leveraging large language models (LLMs) for embodied decision-making (Huang et al., 2022; Raman et al., 2024) and reasoning (Huang et al., 2023), while vision-language models (VLMs) are often considered

to have limited embodied reasoning ability due to their pre-training corpora that focus primarily on language generation (Valmeekam et al., 2023). Common ways of addressing this issue include fine-tuning on datasets from a specific domains (Hong et al., 2023; Mu et al., 2024; Chen et al., 2024) or knowledge distillation (Sumers et al., 2023; Yang et al., 2024). Meanwhile, many works manage to leverage preexisting models without further training from direct visual observation (Fang et al., 2024; Nasiriany et al., 2024) to complete robotic tasks (Jiang et al., 2025). In these works, the embodied reasoning ability of the foundation models serves as the central part of the systems. However, most benchmarking works evaluate the embodied reasoning ability of the models in a question-answering fashion (Sermanet et al., 2024; Majumdar et al., 2024; Cheng et al., 2024; Chen et al., 2025), where it remains unclear whether they are capable of solving robotic tasks.

K Glossary of Notation

We summarize the notation used in the main paper. Modifiers apply throughout; grounded, indexed, and estimated variants are not listed separately unless they require an additional definition.

Notation	Meaning
Modifiers	
$\underline{x}$	Grounded instance; underlining denotes its valid groundings.
$\boldsymbol{x}$	Ordered tuple or sequence, including object arguments, parameters, and plans.
$x^{\uparrow}$	Symbolic abstraction of a state or state space. In result tables.
$\widehat{x}$	Hypothetical or predicted quantity, as specified locally.
x^{*}	The target model or target success ratio.
x'	Successor state.
x^{+}, x^{-}	Add and delete components of an effect.
x_{ω}, x_{a}, x_{p}	Association with a skill, operator, or predicate.
Environment and skills	
$\mathcal{S}, s$	Low-level state space and a state.
$\mathsf{T}, \mathtt{t}(\cdot)$	Object-type set and typing function: $\mathtt{t}(o)$ gives an object's types; $\mathtt{t}(\theta^{i})$ gives a parameter's required type.
$\mathcal{O}, o$	Objects in a scene and an individual object.
Ω, ω	Library of black-box skill schemas and an individual skill.
$\boldsymbol{\theta}$	Typed parameter tuple of a skill, predicate, or operator.
π	Low-level policy of a skill.
$\mathcal{I}_{\omega}$	Pre-success set: states from which the skill succeeds, parameterized by object arguments.
β_{ω}	Success set: states satisfying the skill's successful outcome, parameterized by object arguments.
T	Deterministic low-level transition function.
s_0	The initial state in a skill planning problem.
$\mathcal{S}_g$	Set of goal states in a skill planning problem.
N	Number of scenes in the learning problem.
Symbolic model	
$\mathcal{P}, p$	Predicate vocabulary and a predicate schema.
ℓ_p, C_p	Predicate's semantic label and classifier.
α	Abstraction function mapping a low-level state to its true grounded predicates.
Γ	Grounding function mapping a symbolic condition to the low-level states satisfying it.
$\mathcal{A}, a$	Set of operator schemas and an individual operator.
$\mathrm{Pre}, \mathrm{Eff}$	Operator preconditions and effects; effects have add and delete components.
$\mathcal{M}$	Abstract transition model consisting of predicates and operators: $\langle \mathcal{P}, \mathcal{A} \rangle$.
$\Phi_{\omega}, \Xi_{\omega}$	Model-estimated pre-success and success sets, respectively; instance-specific sets use a grounded skill subscript.

Notation	Meaning (continued)
Λ_o	Lifting relative to skill arguments: retain relevant predicates and replace objects with typed variables.
$\Delta(s, s')$	Abstract change: the pair of added and deleted grounded predicates between two states.
Data collection	
τ	Transition $\langle s, \underline{\omega}, s' \rangle$ recording a skill execution.
$\mathcal{D}$	Collected transition dataset; D denotes its size and $d \subseteq \mathcal{D}_\omega$ an effect cluster.
b, c	Balance and coverage scores for a proposed skill sequence.
r_{suc}	Success ratio; a hat denotes its prediction and a star its target.
Q	Count matrix of consecutive skill pairs; $Q[i, j]$ counts pair (i, j).
$p_{i,j}, H(Q)$	Normalized frequency of skill pair (i, j) and entropy of the skill-pair distribution.
Theoretical analysis	
$\mathcal{H}$	Fixed finite hypothesis class of candidate symbolic models.
$\ell_{\mathcal{M}}(\tau)$	Indicator of disagreement between predicted and true initial or terminal set membership.
d_{compl}	Expected error averaged over transition positions within an episode.
$d_{\text{compl},K}$	Average conditional error across K episodes, conditioning each episode on its preceding history.
n, K	Transitions per episode and number of collection episodes, respectively.
r, t	Episode and within-episode position indices; $\tau_{r,t}$ identifies a transition.
$\mathcal{D}_{1:r-1}$	Collection history preceding episode r.
k, m	Separation lag and effective sample size $m = \lfloor n/k \rfloor$.
$\beta(k)$	Bound on temporal dependence at lag k, distinct from the skill success set β_ω.
ϵ	Allowed expected-error tolerance.
$\ominus$	Symmetric difference of sets.

L AI Usage Disclosure

In this work, we used generative AI tools for formulating mathematical claims, helping develop theoretical models or conceptual frameworks, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs. We have not used generative AI tools for designing or providing feedback on research methodology or experiments, proposing or refining hypotheses, implementing methods, supporting qualitative and thematic data analysis, interpreting results, and generating synthetic data sets, assisting with translation, cleaning and reformatting dataset are not applicable to this work. Additionally, we used generative AI tools for sourcing/searching for information. We have reviewed all AI-assisted work. We proofread the LLM-generated math formulation and proof. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.