SKILLWRAPPER: Skill Abstraction in the Era of Foundation Models

Shreyas Sundara Raman[1*], Ziyi Yang[1*], Benned Hedegaard[1], Stefanie Tellex[1], David Paulius[1†], Naman Shah[1†]

***Abstract*—We envision a future where robots will be equipped "out of the box" with composable and portable skills. However, the conditions in which these skills will successfully execute are not formalized in a way that lets robots autonomously compose skills, posing difficulties for robot programmers who operate such robots. *Abstractions* are a key requirement to enable robots to perform complex tasks. Often, domain experts hand-craft these abstractions, introducing bias from human intuition. Alternatively, computational approaches can be used to invent abstractions autonomously, but human programmers or end users may not be able to interpret the resulting abstractions. We present an approach for autonomously learning natural-language-interpretable abstractions. Our novel method learns symbolic representations for black-box robot skills, such as `GoTo` and `PickUp`, from a high-dimensional and unstructured input in the form of 2D images. Specifically, we use foundation models to propose exploratory sequences of skill executions, to invent symbolic predicates that disambiguate low-level state transitions, and to classify when said predicates hold in a given state. We present preliminary results in a simulated setting, demonstrating the feasibility of our method.**

I. INTRODUCTION

As robotics researchers, we envision a future where robots are deployed to the world to carry out complex, goal-directed tasks, such as arranging a room or delivering items to different locations. Many of such robots will be equipped with numerous pre-designed skills that can be used out of the box. These skills typically involve high-level behaviors such as `GoTo(location)`, `Pickup(item)`, and `Open(door)`. Robots may need to chain or sequence its skills to solve complex tasks, which can be achieved via automated planning [47, 14]. However, planning requires expert-crafted characterizations of robot skills, which can be used to identify under which conditions skills would be successful. In practice, this would require robot manufacturers to provide such information for every deployment of such robots.

Motivated by this, we aim to learn skill abstractions that can enable robot programmers to effectively use out-of-the-box skills. Prior work has shown promising results in learning symbolic representations for high-level skills [28, 29, 51, 48, 18]. However, they assume access to privileged information such as object poses [48] or extensive human feedback [18] while also lacking support for unstructured input and interpretable representations [29]. To facilitate skill abstraction learning across modalities, we look to foundation models. Foundation models, such as large language models (LLMs) and vision-language models (VLMs), have been shown to excel at tasks such as natural language generation, understanding, and image generation. More recently, these models have been applied to robotic settings as a high-level planner with characterized affordance of primitive skills [1, 9, 33], while they are incapable of planning with uncharacterized black-box skills. Instead, we believe foundational models can be helpful in systematically discovering symbolic representations for this purpose.

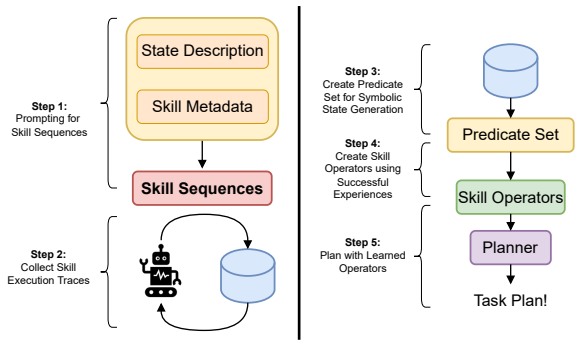

Fig. 1: Brief overview of the SKILLWRAPPER system.

We present SKILLWRAPPER, the first known approach for *autonomously characterizing robot skills using foundation models that provide human interpretability of the learned representations*. We develop a novel method that takes predefined robot skills as input and learns a symbolic robot skill model represented as PDDL [37]. Unlike existing approaches that learn symbolic representations from well-structured state representations, our method is designed to work with high-dimensional and unstructured inputs and generate human-interpretable symbolic representations. In contrast to existing approaches that directly use foundation models for task planning [1, 9, 33, 61, 18], our aim is to develop a system that automatically characterizes black-box skills from unstructured input representation and enables us to leverage off-the-shelf automated planners [19].

Our system exploits foundation models in three key ways. First, we leverage them as zero-shot classifiers for truth values of predicates given image inputs. Second, we develop a novel active data collection algorithm that uses a foundation model to propose skill sequences for the robot, enabling our method to generate contrastive examples of executing skills. Lastly, we develop a novel algorithm to discover human-interpretable symbolic *predicates* and their semantics with foundation models. We perform preliminary experiments that evaluate our system in simulated settings that require complex long-horizon reasoning and composition of skills to complete the tasks. Empirical results show that our method efficiently uses the foundation model in each of the aforementioned modules and learns usable and interpretable symbolic representations for black-box skills.

Project website: https://yzylmc.github.io/skill-wrapper/

[1]Brown University, Providence, RI, USA.

*Equal contribution, listed in alphabetical order. Corresponding Author (*Email*: ziyi_yang1@cs.brown.edu)

†Equal advising, listed in alphabetical order.

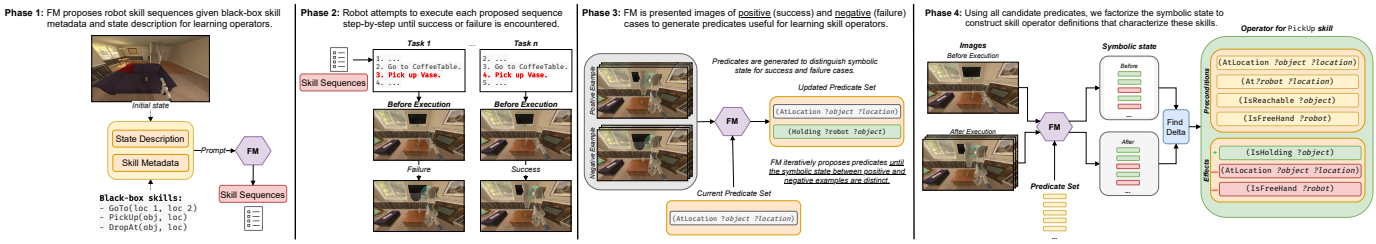

Fig. 2: Overview of SKILLWRAPPER, which is comprised of four main phases for learning operators: 1) we use a foundation model (FM) to propose sequences of skill executions useful to operator learning given a description of the agent's environment and metadata about its skills; 2) the robot attempts to execute each proposed sequence, while capturing the initial and final state after each action is performed and storing these execution traces in a dataset; 3) we review the dataset of experiences collected by the robot and present them to the FM as contrastive pairs (i.e., success and failure images as positive and negative examples), from which a FM will propose a set of predicates suitable to describe a symbolic state across all skills; and 4) using the predicate set and a pair of images corresponding to a positive example are given to a FM to infer the symbolic state before and after successful execution, which are then used to define skill operators.

II. Related Work

The core focus of our work is to use pre-trained foundation models to learn human-interpretable symbolic representations of black-box robot skills useful to downstream decision-making. This work draws ideas from different fields of research such as model learning, abstraction learning, and task and motion planning (TAMP). Foundation models have attracted a lot of attention in recent years for robotics. Several methods have used foundation models (mostly LLMs) as high-level planners [1, 44, 9]. Several approaches have used foundation models as robot action models [4, 50] or to generate reward functions for robot tasks [59]. While these approaches show promising results for short-horizon single-skill problems, they fail to scale to complex long-horizon problems. Lastly, multiple approaches have used foundation models to learn symbolic representations for robot skills, but they require extensive human expert feedback [18]. To the best of our knowledge, our work is the first to use a foundation model in different modules to automatically learn the human-interpretable symbolic characterization of robot skills.

TAMP has been long used for solving complex robot tasks [8, 47, 14]. However, these approaches require symbolic models for the robot skills that they can use to plan robots' actions. Various approaches [29, 51, 48] have been developed to learn such symbolic models compatible with TAMP solvers, but these approaches cannot handle high-dimensional or unstructured input, such as image observations or natural language. Additionally, abstractions learned through these methods are not human interpretable. On the other hand, we explicitly design our approach to work with high-dimensional inputs and generate human-interpretable abstractions using pre-trained foundation models. The proposed system also connected to other domains in robotics, and the full related work can be found in Appendix VIII-G.

III. Formal Framework

We consider an active learning setting where an agent is provided a collection of pre-defined, "black-box" skills. However, because the agent is not provided a transition model describing the skills' dynamics, it cannot easily sequence the skills to achieve its goals. We propose a method that learns one such transition model in the form of symbolic PDDL-style [37] predicates and operators. To learn this model, the method commands the agent to execute sequences of skills, creating a dataset of observed environment transitions. This method must then iteratively update its learned model based on the available data, propose additional skill sequences to generate further data, and accordingly improve the learned model. Once the method has learned an appropriate abstract transition model, an agent can use off-the-shelf classical AI planners to sequence its skills to solve unseen, long-horizon problems.

A. Environment Model

We model the agent's environment as a semi-Markov decision process (SMDP) [3] defined by the tuple $(\mathcal{S}, \Omega, R, T, \gamma)$, extended by the tuple $(\mathcal{T}, \mathcal{O})$. The state space $\mathcal{S}$ is assumed to be high-dimensional, continuous, and fully observable. The agent encounters a fixed set of known, typed objects $o \in \mathcal{O}$ in the environment. Each object's type, denoted $\text{type}(o)$, is drawn from the known set of types $\mathcal{T}$. The agent is provided a set of "black-box" skills Ω, modeled as options [53] taking discrete, typed parameters (see Sec. III-B).

The reward function $R(s' \mid s, \omega)$ and transition function $T(s' \mid s, \omega)$ respectively describe the reward for, and probability of, arriving in the state $s' \in \mathcal{S}$ after executing the skill $\omega \in \Omega$ from the state $s \in \mathcal{S}$. This paper does not consider explicit time steps when modeling R or T. The discount factor $\gamma \in (0, 1]$ describes the agent's preference for short-term or long-term reward. The reward model R and transition model T for a particular skill $\omega \in \Omega$ are called the skill's model. Because we do not assume access to skill models, our method must learn some alternative model of the skills' dynamics.

B. Black-Box Skills

We model the agent's skills Ω as options [53] taking discrete parameters with types drawn from the set $\mathcal{T}$. When the agent

executes one of its skills $\omega \in \Omega$, it enters a temporally extended course of action, traversing multiple states $s \in \mathcal{S}$ as the agent takes many atomic actions $a \in \mathcal{A}$. Formally, each skill $\omega \in \Omega$ is defined by the tuple $(I_\omega, \pi_\omega, \beta_\omega, \boldsymbol{\theta}_\omega)$, where an initiation set $I_\omega \subseteq \mathcal{S}$, which we defines the states where the skill can be executed, a deterministic policy $\pi_\omega : \mathcal{S} \to \mathcal{A}$ selects atomic actions during the skill, a termination condition $\beta_\omega : \mathcal{S} \to [0, 1]$ terminates the skill with probability $\beta_\omega(s)$, and a tuple of discrete, typed parameters $\boldsymbol{\theta}_\omega = (\theta^1_\omega, \ldots, \theta^n_\omega)$ stand in for objects of specific types, such that $\text{type}(\theta^i_\omega) \in \mathcal{T}$. We assume the agent knows the skills' typed parameters $\boldsymbol{\theta}_\omega$. However, the agent can only evaluate skill applicability (i.e., whether $s \in I_\omega$) for the state it is currently in.

C. Skill Abstraction

Without full knowledge of the skill models and initiation sets, the agent needs some method to learn a suitable transition model for its pre-defined skills. This method must command the agent to execute sequences of skills, creating a dataset of observed environment transitions that can be used to learn a model of the skills' dynamics. We assume that the learned transition model consists of PDDL-style [37] predicates and operators. If learned successfully, such abstractions would enable the agent to use an off-the-shelf classical planner to compose its skills to accomplish unseen, long-horizon goals.

This learning process invents predicate symbols $\mathcal{P}$ to express abstract relations between objects $\mathcal{O}$ in the environment. Formally, a predicate $P \in \mathcal{P}$ of arity $n \in \mathbb{N}_1$ is defined by the tuple $(\boldsymbol{\theta}_P, c_P)$, where $\boldsymbol{\theta}_P = (\theta^1_P, \ldots, \theta^n_P)$ is a tuple of typed parameters such that $\text{type}(\theta^i_P) \in \mathcal{T}$ and $c_P : \mathcal{O}^n \to \mathcal{S} \to \{0, 1\}$ is a lifted Boolean state classifier.

Using the predicates $\mathcal{P}$, the learning method must invent a set of operators (i.e., relational abstract actions) defining abstract transition models for the skills $\omega \in \Omega$. Each operator $\alpha \in \overline{\mathcal{A}}$ is defined by the tuple $(\omega_\alpha, \textsc{Pre}_\alpha, \textsc{Eff}^+_\alpha, \textsc{Eff}^-_\alpha, \boldsymbol{\theta}_\alpha)$, where $\omega_\alpha \in \Omega$ is the skill corresponding to the operator; preconditions $\textsc{Pre}_\alpha \subseteq \mathcal{P}$ define the abstract conditions necessary to apply the operator; add effects $\textsc{Eff}^+_\alpha \subseteq \mathcal{P}$ and delete effects $\textsc{Eff}^-_\alpha \subseteq \mathcal{P}$ describe the abstract conditions that become true and false, respectively, after the operator is applied; and $\boldsymbol{\theta}_\alpha = (\theta^1_\alpha, \ldots, \theta^n_\alpha)$, with $\text{type}(\theta^i_\alpha) \in \mathcal{T}$, is a tuple of discrete, typed parameters defining placeholders for objects $o \in \mathcal{O}$.

D. Grounded Abstractions

Before the learning method can evaluate whether a predicate $P \in \mathcal{P}$ of arity $n \in \mathbb{N}_1$ holds in a given state, the predicate must first be grounded by binding its parameters $\boldsymbol{\theta}_P$ to particular objects $o_{1:n}$ in the world. The abstract relation described by c_P can then be evaluated for those objects in a particular state. Formally, a grounded predicate $\underline{P} = P(o_1, \ldots, o_n)$ is defined if and only if $\forall i \in \{1, \ldots, n\}, \text{type}(\theta^i_P) = \text{type}(o_i)$. If $\underline{P}$ is defined, the grounding process also induces the Boolean state classifier $c_{\underline{P}} : \mathcal{S} \to \{0, 1\}$, where $c_{\underline{P}} = c_P(o_1, \ldots, o_n)$ represents a truth-valued relation between the objects $o_{1:n} \in \mathcal{O}^n$.

Together, objects $\mathcal{O}$ and predicates $\mathcal{P}$ induce a grounded predicate set $\underline{\mathcal{P}}$ containing all expressible relations between objects. We define $\underline{\mathcal{P}}$ as the set of all valid groundings of the predicates $P \in \mathcal{P}$, each of arity $n \in \mathbb{N}_1$, using objects $o_i \in \mathcal{O}$:

$$\underline{\mathcal{P}} = \left\{P(o_{1:n}) \mid \forall i \in \{1, \ldots, n\}, \text{type}(\theta^i_P) = \text{type}(o_i)\right\}. \quad (1)$$

We define the abstract state space $\underline{\mathcal{S}} = 2^{\underline{\mathcal{P}}}$ as the power set of $\underline{\mathcal{P}}$ so that each abstract state corresponds to a specific combination of the possible grounded relations. An abstraction function $F_{\mathcal{P}} : \mathcal{S} \to \underline{\mathcal{S}}$ maps each low-level state $s \in \mathcal{S}$ to an abstract state $\underline{s} \in \underline{\mathcal{S}}$ corresponding to the subset of grounded predicates $\underline{P} \in \underline{\mathcal{P}}$ that are true in state s.

$$\underline{s} = F_{\mathcal{P}}(s) \triangleq \{\underline{P} \mid \underline{P} \in \underline{\mathcal{P}} \wedge c_{\underline{P}}(s) = 1\}. \quad (2)$$

The abstract action space $\underline{\mathcal{A}}$ is defined similarly by grounding the operators $\alpha \in \overline{\mathcal{A}}$. Each grounded operator $\underline{\alpha} \in \underline{\mathcal{A}}$ is defined by a tuple $(\alpha, o_{1:n})$, where $\alpha \in \overline{A}$ is an operator with $n \in \mathbb{N}_1$ typed parameters $\boldsymbol{\theta}_\alpha = (\theta^1_\alpha, \ldots, \theta^n_\alpha)$ and $o_{1:n} \in \mathcal{O}^n$ is a tuple of objects such that $\forall i \in \{1, \ldots, n\}, \text{type}(\boldsymbol{\theta}^i_\alpha) = \text{type}(o_i)$. When these types match, each object argument o_i can be bound to the corresponding parameter θ^i_α, thereby grounding the predicates of the operator α and inducing the ground preconditions $\textsc{Pre}_{\underline{\alpha}} \subseteq \underline{\mathcal{P}}$, ground add effects $\textsc{Eff}^+_{\underline{\alpha}} \subseteq \underline{\mathcal{P}}$, and ground delete effects $\textsc{Eff}^-_{\underline{\alpha}} \subseteq \underline{\mathcal{P}}$.

E. Learning Abstractions from Experience

To learn appropriate predicates and operators, the method commands the agent to execute its skills in the world and gather transition data. Each command is a sequence of skills $\sigma = [\omega_1(\boldsymbol{o}_1), \ldots, \omega_m(\boldsymbol{o}_m)]$, where $\omega_j \in \Omega$, $|\boldsymbol{\theta}_{\omega_j}| = |\boldsymbol{o}_j|$, $\boldsymbol{o}_j = (o^1_j, \ldots, o^n_j) \in \mathcal{O}^n$, and $\forall i \in \{1, \ldots, n\}, \text{type}(\theta^i_{\omega_j}) = \text{type}(o^i_j)$. By executing σ, the agent collects a dataset $\mathcal{D}$ of transitions of the form $\tau = (s, \omega(\boldsymbol{o}), s')$, where $s, s' \in \mathcal{S}$, $\omega \in \Omega$, and $\boldsymbol{o} \in \mathcal{O}^n$. These transitions provide two possible pieces of information:

1) Whether the skill ω can be initiated from the state s, given the object arguments $\boldsymbol{o}$.
2) If $s \neq s'$, the difference in the abstract states $\underline{s} = F_{\mathcal{P}}(s)$ and $\underline{s'} = F_{\mathcal{P}}(s')$ should correspond to the effects of some grounded operator $\underline{\alpha} \in \underline{\mathcal{A}}$ modeling the skill ω.

F. Problem Definition

Definition 1. *Given a set of skills Ω, an SMDP $(\mathcal{S}, \Omega, R, T, \gamma)$, and a set of objects $\mathcal{O}$ of types $\mathcal{T}$, we define an* Active Relational Abstraction Learning Problem *as learning an abstract transition model $\mathcal{M} = (\mathcal{P}, \overline{\mathcal{A}})$ for the skills Ω.*

IV. Method

Briefly, SkillWrapper characterizes skills from a library of black-box robot skills by iteratively proposing and executing exploratory skill sequences to collect experiences in an active way (as described in Section III-E). Using a foundation model, our system progressively invents new predicates to construct a predicate set $\mathcal{P}$, which are then used to construct symbolic operators for each skill. To construct a sufficient descriptor set for a symbolic state, which are discussed in predicate invention, skill sequence proposal, and operator

learning; the overall process is illustrated in Algorithm 1. The resulting wrapped skills, or the operators learned by our system, can be chained together to solve task planning problems specified with natural language or images.

Algorithm 1 SKILLWRAPPER

Input: Set of skills Ω, number of iterations $m \in \mathbb{N}_1$
Output: Abstract transition model $\mathcal{M} = (\mathcal{P}, \overline{\mathcal{A}})$
$\mathcal{D}, \mathcal{P}, \overline{\mathcal{A}} \leftarrow \emptyset$
for $i \in \{1, \ldots, m\}$ **do**
    $\sigma \leftarrow$ PROPOSESKILLSEQUENCES$(\Omega, \mathcal{P}, \mathcal{D})$
    $\mathcal{D} \leftarrow \mathcal{D} \cup$ EXECUTESKILLS(σ)
    $\mathcal{P} \leftarrow$ INVENTPREDICATES$(\Omega, \mathcal{D}, \mathcal{P})$
    $\overline{\mathcal{A}} =$ LEARNOPERATORS$(\mathcal{T}, F)$
end for
return $\mathcal{M}$

A. Predicate Invention

Predicates are the basic units of our skill representation. To generate interpretable predicates that facilitate task specification, our system leverages foundation models to propose predicates together with their semantic meanings in English sentences. Specifically, our system uses a foundation model to propose skill sequences that help the agent gather experiences of skill execution, from which our system then progressively invents predicates that disambiguate successful and failed executions. Foundation models also enable state abstraction from visual observations in our system: since the predicates are interpretable, foundation models essentially function as zero-shot classifiers to determine the truth value of each predicate.

The objective of predicate invention is to construct a set of predicates via prompting to describe important factors relevant to skill executions in abstract states, and the ability to describe such abstract states can be validated via the ability to distinguish underlying world states. For example, suppose two executions of the same skill from the same abstract state lead to different execution successes (where one failed and the other succeeded). In that case, it implies that the symbolic representation captured by the existing predicate set is insufficient to describe the difference in skill execution success, and additional predicates need to be invented in order to disambiguate the abstract representations of both starting states. To trigger new predicate invention, the algorithm explicitly looks for such mismatched pairs from the collected dataset of skill execution traces: for preconditions, comparisons are conducted between the states before skill executions, and for effects, comparisons are conducted between the truth value changes of predicates before and after skill execution.

When inventing a new predicate, our system prompts a foundation model with the mismatched pair, metadata of the skill including its high-level function and argument types, and the existing predicate set and its associated semantics in English sentences. The new predicate will be proposed together with its semantics, and they are expected to distinguish the mismatched pair and be semantically different from existing predicates. The invented predicate is then evaluated over the collected skill execution traces using two metrics regarding precondition and effect, respectively (the scoring function can be found in Appendix VIII-A.) The predicate invention process is demonstrated in Algorithm 2.

Algorithm 2 Invent Predicates

Input: Predicate set $\mathcal{P}$, Skill set Ω, skill executability $I_\omega(s)$, predicate classifier $F_\mathcal{P}(s)$, and skill executions traces $\mathcal{D}(\omega) = \{(s, s')\}_\omega$.
Output: Predicate set $\mathcal{P}$
for ω in Ω **do**
    while $\exists (s_i, s_i'), (s_j, s_j') \in \mathcal{D}(\omega)$ **such that** $F_\mathcal{P}(s_i) = F_\mathcal{P}(s_j)$ **and** $I_\omega(s_i) \neq I_\omega(s_j)$ **do**
        $P \leftarrow new\_predicate$
        $\mathcal{P} \leftarrow \mathcal{P} \cup P$ **if** VALIDATEPRECOND$(P, \omega, \mathcal{D})$ = True
    end while
    while $\exists (s_i, s_i'), (s_j, s_j') \in \mathcal{D}(\omega)$ **such that** $F_\mathcal{P}(s_i') - F_\mathcal{P}(s_i) = F_\mathcal{P}(s_j') - F_\mathcal{P}(s_j)$ **and** $I_\omega(s_i) \neq I_\omega(s_j)$ **do**
        $P \leftarrow new\_predicate$
        $\mathcal{P} \leftarrow \mathcal{P} \cup P$ **if** VALIDATEEFF$(P, \omega, \mathcal{D})$ = True
    end while
end for
return $\mathcal{P}$

B. Skill Sequence Proposal

Our system queries foundation models to generate skill sequences in natural language that support predicate invention for black-box skills by exploring the symbolic state space in a directed manner. LLMs are useful for skill sequence proposal due to their flexible interface that can infer skills and predicates defined in natural language within the prompts, and their ability to propose sequences that abide multiple complex constraints.

Rather than naively sampling skill sequences from the LLM's token distribution, the predicate invention algorithm benefits more from proposed skill sequences that balance three important constraints, shown algorithmically in Section VIII-B:

a) Coverage (C):

$$C = \frac{\mathcal{Q}'}{\Sigma \mathcal{Q}'} \times log(\frac{\mathcal{Q}'}{\Sigma \mathcal{Q}'}) - \frac{\mathcal{Q}}{\Sigma \mathcal{Q}} \times log(\frac{\mathcal{Q}}{\Sigma \mathcal{Q}}) \tag{3}$$

Coverage score evaluates the information gain on all possible skill pairs (two skills executed consecutively) over existing skill execution traces by executing a new skill sequence. Specifically, the information gain is measured by the increase in Shannon entropy [49] over the distribution of all skill pairs resulting from executing the proposed skill sequence as shown in Equation 3, where $\mathcal{Q}$ is a table counting the number of executed skill pair before executing the proposed sequence and $\mathcal{Q}'$ is a table for after executing the sequence. Maximizing coverage encourages the proposed skill sequence to contain least explored skill pairs and uncovers a larger set of inter-dependencies across the preconditions and effects of all skills.

b) Chainability (Ch): Chainability measures the ratio of successful to failed skill executions added to the dataset of skill execution traces due to executing a proposed skill sequence. By computing chainability, we estimate the degree to which the preconditions and effects of learned operators at each iteration are satisfied, and executability can be inferred from the estimated symbolic states and the operators. With an appropriate chainability score, the collected dataset of skill execution traces maintains a balance between number of successful executions and failure executions, which is ideal for identifying possible mismatched pairs and thus inventing predicates.

c) Consistency (Co):

$$Co = \sum_{s \in stateseq} -log(KDE(s, \mathcal{D})) \tag{4}$$

Consistency estimates the probability density that the lifted symbolic states resulting from executing skills in the proposed sequence, equals certain lifted symbolic states in the dataset of skill execution traces $\mathcal{D}$, i.e., the probability that $F_{\mathcal{P}}(s_i) = F_{\mathcal{P}}(s_j)$, where $s_i \in \sigma$ and $s_j \in \mathcal{D}$. A kernel density estimator (KDE) [28][43] with the Hamming distance kernel [17] is used to estimate this probability density. Maximizing this metric prioritizes skill sequences that with the highest likelihood that the resulting lifted symbolic states match the symbolic states in the dataset of execution traces, thus increasing the chance of encountering a mismatched pair with identical symbolic states required for predicate invention.

When proposing skill sequences, the foundation model is provided the skill set Ω and proposed predicate set $\mathcal{P}$, whilst being prompted to deliberately explore the boundary of the predicates' truth values. The skill sequence proposing algorithm assigns a score tuple (C, Ch, Co) for every proposed sequence and maintains a subset of pareto-front sequences that cannot "strictly dominate" another sequence, i.e., $C_i < C_j \vee Ch_i < Ch_j \vee Co_i < Co_j \vee (C_i \leq C_j \wedge Ch_i \leq Ch_j \wedge Co_i \leq Co_j)$ where $i \neq j$; the output skill sequence is chosen from this pareto-front subset.

Algorithm 3 Propose Skill Sequences

Input: Operator set $\overline{\mathcal{A}}$, Dataset of skill execution traces $\mathcal{D}$, batch size of candidate sequences n
Output: Proposed skill sequence σ
seq_batch = GENERATEGROUNDEDSEQUENCES($\overline{\mathcal{A}}, n$)
$Score \leftarrow \{\}$
for σ **in** seq_batch **do**
    $cov \leftarrow$ COVERAGE($\mathcal{D}, \sigma$)
    $chain, stateseq \leftarrow$ CHAINABILITY($\overline{\mathcal{A}}, \mathcal{P}, \sigma$)
    $suff \leftarrow$ CONSISTENCY($\mathcal{D}, stateseq$)
    $Score[\sigma] = (cov, chain, suff, stateseq)$
end for
$\sigma^* =$ PARETOOPTIMALITY($Score, \mathcal{D}, \overline{\mathcal{A}}, \mathcal{P}$)
return σ^*

C. Operator Learning by Clustering

The operator learning algorithm primarily focuses on determining unique effects and preconditions of individual subgoal options from the original skill. Unlike prior work [29], which accesses low-level state variables and constructs symbols with factors bottom-up, SKILLWRAPPER builds symbols directly using language-specified predicates. From the dataset of skill execution traces, the system evaluates the truth value of predicates at every state and thus the effect (truth value change) accordingly and cluster the executions with the same lifted effect change into one operator. After the clustering, the precondition of each operator can be derived by taking the intersection of the predicates' truth values at all states before skill executions associated with the operator. Only successful skill executions from the dataset are used for the operator learning process, since failure cases are not informative regarding precondition or effect of skills.

Algorithm 4 Learn Operators

Input: Skill execution traces $\mathcal{D}(\omega) = \{(s, s')\}_\omega$, predicate classifier $F_{\mathcal{P}}(s)$
Output: Operators set $\overline{\mathcal{A}}$
eff_dict $\leftarrow$ defaultdict() ▷ Store clustered effects
for (s, s') in $\mathcal{D}$ **do**
    $eff = F_{\mathcal{P}}(s') - F_{\mathcal{P}}(s)$
    eff_dict[eff].add((s, s'))
end for
$\overline{\mathcal{A}} \leftarrow$ []
for eff in eff_dict **do**
    $execution\_list \leftarrow$ eff_dict[eff]
    $precond \leftarrow \Pi_{(s,s') \in execution\_list} F_{\mathcal{P}}(s)$
    $\overline{\mathcal{A}}$.add([$precond, eff$])
end for
return $\overline{\mathcal{A}}$

D. Planning with Wrapped Skills

After learning operators from the execution traces, SKILLWRAPPER essentially wraps the original skills by the operators, and it is then capable of solving task planning problems with these wrapped skills. Since the learned predicates are interpretable, human users can conveniently specify tasks in natural language instructions, and the specifications can be converted into PDDL statements in a similar way as existing work [34]. In our work, the burden of the foundation model is reduced to only translating the problem statement, because actions and predicates required for the domain definition can be directly ported from the learned operators. Moreover, in the same way as evaluating truth values of predicates given image inputs, the foundation model also enables specifying initial states and goal states using truth values of predicates from images. After the PDDL solver returns a task plan, our system retrieves the arguments from each operator to make the plan compatible with the original skills for execution.

V. EXPERIMENTS

We demonstrate the capabilities of the SKILLWRAPPER system in a preliminary set of simulated experiments. In our experiments, we prompt GPT-4o [41] with egocentric observations to evaluate the truth values of predicates; for predicate invention and skill sequence proposal, we utilize text-only prompts with o1-preview [42].

A. Experimental Setup

We implemented SKILLWRAPPER in ManipulaThor [27, 10], a simulated indoor environment containing a mobile manipulator, three objects, and three receptacles. The robot's visual observation is egocentric. We designed three high-level actions: `PickUp(obj, loc)`, `DropAt(obj, loc)`, and `GoTo(loc1, loc2)`, each parameterized by objects of particular types in the environment. These high-level actions were provided to the simulated robot as black-box skills that execute a deterministic sequence of the low-level motion actions available to the robot. Each of the skills was designed to have implicit preconditions and effects. For example, the robot cannot execute `PickUp` on a far-away object, and cannot execute `DropAt` when the targeted receptacle is out of reach. When the robot attempted to execute an inapplicable skill, the "failed" execution would result in the same state as before, leaving no effect on the environment state.

B. Task Proposal and Predicate Invention

As outlined in Algorithm 1, SKILLWRAPPER iteratively proposes skill sequences and collects skill execution traces to learn an abstract transition model for the given skills. In our experimental implementation, each loop began with the skill sequence proposal component proposing a sequence of eight skills applied to concrete objects. The simulated robot then executed the skill sequence, obtaining a trace of environment transitions. During execution, the robot captured visual observations before and after executing each skill, forming a dataset of transitions $\mathcal{D} = \{(s, \omega(\boldsymbol{o}), s')\}$ as described in Section III-E. In total, we gathered data from five skill sequences consisting of 40 environment transitions and 80 images as state representations, out of which 24 skill executions were successful and 16 were not. The proposed skill sequences are provided in Appendix VIII-F.

(a) (b)

Fig. 3: Example mismatched pair from the collected dataset of skill execution traces. Both images represent the states before executing `PickUp(Vase, CoffeeTable)`, and they have the same symbolic states {`IsClose(CoffeeTable)`=True}, while (a) succeeded but (b) failed since the gripper is already holding the *Bowl*. The predicate invention algorithm proposes a new predicate `HasEmptyHand()` to distinguish the two states.

The foundation model was then used to classify the truth value of each grounded predicate in every observed state. Given these truth values, the SKILLWRAPPER system iterated over the transitions $\{(s, \omega(\boldsymbol{o}), s')\}$, looking for "mismatched" pairs of transition-initial states $s_i, s_j \in \mathcal{S}$ such that $F_{\mathcal{P}}(s_i) = F_{\mathcal{P}}(s_j)$ (i.e., identical abstract states) yet $I_\omega(s_i) \neq I_\omega(s_j)$ (i.e., the skill was only executable in one of the states). These cases triggered predicate invention, as described in section IV-A.

After five loops of this process (i.e., $m = 5$), the system had generated six unique predicates: $\mathcal{P} = \{$`HasEmptyHand()`, `IsAt(obj, loc)`, `IsClose(loc)`, `IsGrasped(obj)`, `IsLoose(obj)`, `IsUnoccupied(loc)`$\}$.

C. Learning Operators

Given the dataset of environment transitions $\mathcal{D}$ and the abstract state space $\underline{\mathcal{S}}$ induced by the invented predicates $\mathcal{P}$, Algorithm 4 learned ten operators from the original three high-level skills: two operators for `DropAt`, two for `GoTo`, and six for `PickUp`. This result may reflect noise introduced by the unrealistic images from the chosen simulator, which does not produce a visible change when the robot grasps an object. Here we show one representative learned operator for each high-level skill used in our experiments:

```
(:action PickUp1
    (:parameters ?l - location ?o - object)
    (:precondition (and
        (hasemptyhand)
        (isat ?o ?l)
        (isclose ?l)
        (not (isgrasped ?o))
        (isloose ?o)
        (not (isunoccupied ?l)))
    )
    (:effect (and
        (not (hasemptyhand ))
        (not (isat ?o ?l))
        (isgrasped ?o)))
)

(:action DropAt2
    (:parameters ?l - location ?o - object)
    (:precondition (and
        (hasemptyhand)
        (not (isat ?o ?l))
        (isclose ?l)
        (isgrasped ?o)
        (isloose ?o)
        (not (isunoccupied ?l)))
    )
    (:effect (and
        (isat ?o ?l)
        (not (isgrasped ?o)))
    )
)

(:action GoTo1
    (:parameters ?l2 - location ?l1 - location)
    (:precondition (and
        (hasemptyhand)
        (isclose ?l1)
        (not (isclose ?l2)))
    )
    (:effect (and
        (not (isclose ?l1))
        (isclose ?l2))
    )
)
```

D. Planning with Wrapped Skills

As detailed in Section IV-D, the interpretable predicates produced by our system conveniently enable task specification using natural language or images. A foundation model was used to convert these human-interpretable task specification formats into a task planning problem of the form $(\underline{s}_0, \underline{\mathcal{S}}_g)$,

consisting of an initial abstract state $\underline{s}_0 \in \underline{\mathcal{S}}$ and a set of goal conditions $\underline{\mathcal{S}}_g \subseteq \underline{\mathcal{P}}$. Our experiments used the Fast Downward [19] planner for task-level planning. In the presented experiments, images were only used to specify tasks' initial states, representing a situation where the robot does not have access to visual observations of goal states before execution.

Here we present an example of multi-modal task specification using natural language, specifically English, and images. The natural language instruction describes the name of each object, the location of the robot, and the goal conditions. The linguistic description of the robot's location implicitly conveys the truth value of the grounded predicate `IsClose(Sofa)`. The truth values of all other grounded predicates in the initial abstract state are inferred from the images.

"There are three items *Vase*, *TissueBox*, and *Bowl*, and three locations *Sofa*, *CoffeeTable*, and *DiningTable*. Their initial positions are shown as follows. The robot is near the *Sofa* initially, and everything is placed stable, and all items can fit in every location. The goal is to have all items on the Sofa."

Fig. 4: Example of multi-modal task specification using natural language and egocentric visual observations.

Provided the learned abstract transition model $\mathcal{M} = (\mathcal{P}, \overline{\mathcal{A}})$, and the task planning problem $(\underline{s}_0, \underline{\mathcal{S}}_g)$ inferred from the above task specification, the task planner returned the plan:

```
[
    GoTo3(Sofa,CoffeeTable),
    PickUp5(Vase,CoffeeTable),
    GoTo2(CoffeeTable, DiningTable),
    DropAt2(Vase,Sofa),
    GoTo4(Sofa,DiningTable),
    PickUp1(Bowl,DiningTable),
    GoTo2(DiningTable,Sofa),
    DropAt2(Bowl,Sofa)
]
```

After these operators were converted to the corresponding skills, the resulting skill sequence was executed by the simulated robot. The final state satisfied the given task specification as shown in Figure 5. The inferred task planning problem and the learned abstract transition model, both specified as PDDL, can be found in Appendix VIII-E and VIII-D.

VI. Discussion

As the embodied reasoning capabilities of foundation models improve over time, our system may benefit by replacing its backend with newer, more powerful models. We observed this improvement qualitatively when switching from gpt-4-0613 to GPT-4o. A major limitation of our current approach

Fig. 5: Egocentric view of the environment state after executing the sequence of skills corresponding to the generated plan. The final state satisfies the language-specified goal conditions.

is a constraint between the arguments of the given skills and learned predicates. Because we constrain the arguments of invented predicates to a subset of the arguments of the relevant skill, our system can fail to represent certain types of "argument-implicit" conditional effects. For example, given the skill `GoTo(loc)`, our system would not be able to invent a predicate such as `Nearby(obj)` to model effects such as when `GoTo` brings the robot near an unmentioned object. Another limitation of our approach concerns the inherent uncertainty caused by the robot's egocentric view. Without additional information, as formulated in this paper, the agent may not be able to determine truth values for all grounded predicates due to partial observability.

VII. Conclusion

In this paper, we formulate the problem of actively learning interpretable abstract representations of black-box skills. We propose the SkillWrapper system as a solution building upon the multi-modal reasoning capabilities of foundation models. We demonstrate our approach using a proof-of-concept example in a simulated mobile manipulation setting. In ongoing and future work, we plan to evaluate our approach in comparison with alternative methods for active relational abstraction learning, formulate theoretical guarantees characterizing the abstractions learned by our system, and address the limitations mentioned in the discussion section.

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 287–318. PMLR, 14–18 Dec 2022.

[2] Michael Ahn, Debidatta Dwibedi, Chelsea Finn, Montserrat Gonzalez Arenas, Keerthana Gopalakrishnan, Karol Hausman, Brian Ichter, Alex Irpan, Nikhil J Joshi, Ryan Julian, Sean Kirmani, Isabel Leal, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, sharath maddineni, Kanishka Rao, Dorsa Sadigh, Pannag R Sanketi, Pierre Sermanet, Quan Vuong,

Stefan Welker, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, and Zhuo Xu. AutoRT: Embodied Foundation Models for Large Scale Orchestration of Robotic Agents. In *First Workshop on Vision-Language Models for Navigation and Manipulation (VLMNM) at ICRA 2024*, 2024.

[3] Steven Bradtke and Michael Duff. Reinforcement learning methods for continuous-time markov decision problems. *Advances in neural information processing systems*, 7, 1994.

[4] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alex Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. In *arXiv preprint arXiv:2307.15818*, 2023.

[5] Boyuan Chen, Zhuo Xu, Sean Kirmani, Brain 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*, pages 14455–14465, 2024.

[6] William Chen, Oier Mees, Aviral Kumar, and Sergey Levine. Vision-Language Models Provide Promptable Representations for Reinforcement Learning. *arXiv preprint arXiv:2402.02651*, 2024.

[7] 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 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14291–14302, 2024.

[8] 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.

[9] 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*, volume 202 of *Proceedings of Machine Learning Research*, pages 8469–8488. PMLR, 23–29 Jul 2023.

[10] Kiana Ehsani, Winson Han, Alvaro Herrasti, Eli VanderBilt, Luca Weihs, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. ManipulaTHOR: A Framework for Visual Object Manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4497–4506, 2021.

[11] Kuan Fang, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Adaptive Procedural Task Generation for Hard-Exploration Problems. In *International Conference on Learning Representations*, 2021.

[12] 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. *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8, 2022.

[13] Kuan Fang, Fangchen Liu, Pieter Abbeel, and Sergey Levine. MOKA: Open-World Robotic Manipulation through Mark-Based Visual Prompting. *Robotics: Science and Systems (RSS)*, 2024.

[14] 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.

[15] Jiayuan Gu, Devendra Singh Chaplot, Hao Su, and Jitendra Malik. Multi-skill Mobile Manipulation for Object Rearrangement. In *The Eleventh International Conference on Learning Representations (ICML)*, 2022.

[16] Huy Ha, Pete Florence, and Shuran Song. Scaling Up and Distilling Down: Language-Guided Robot Skill Acquisition. In Jie Tan, Marc Toussaint, and Kourosh Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 3766–3777. PMLR, 06–09 Nov 2023.

[17] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[18] 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 *Robotics: Science and Systems (RSS)*, 2024.

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

[20] 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 A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 20482–20494. Curran Associates, Inc., 2023.

[21] 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*, volume 162 of *Proceedings of Machine Learning Research*, pages 9118–9147, 2022.

[22] 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*, volume 205 of *Proceedings of Machine Learning Research*, pages 1769–1782. PMLR, 14–18 Dec 2023.

[23] 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. *Conference on Robot Learning (CoRL)*, 2024.

[24] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized Level Replay. In *International Conference on Machine Learning*, pages 4940–4950. PMLR, 2021.

[25] 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*, pages 2901–2910, 2017.

[26] Brendan Juba, Hai S Le, and Roni Stern. Safe learning of lifted action models. In *Proc. KR*, volume 18, pages 379–389, 2021.

[27] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017.

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

[29] 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.

[30] Leonardo Lamanna, Luciano Serafini, Mohamadreza Faridghasemnia, Alessandro Saffiotti, Alessandro Saetti, Alfonso Gerevini, and Paolo Traverso. Planning for learning object properties. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI Press, 2023.

[31] Amber Li and Tom Silver. Embodied Active Learning of Relational State Abstractions for Bilevel Planning. In *Conference on Lifelong Learning Agents (CoLLAs)*, 2023.

[32] 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.

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

[34] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering Large Language Models

with Optimal Planning Proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

[35] 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, Sasha Sax, and Aravind Rajeswaran. OpenEQA: Embodied Question Answering in the Era of Foundation Models. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[36] 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 *International Conference on Learning Representations*. International Conference on Learning Representations, ICLR, 2019.

[37] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL – The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[38] Toki Migimatsu and Jeannette Bohg. Grounding Predicates through Actions. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3498–3504, 2022.

[39] 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*, volume 36, 2024.

[40] 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 *Forty-first International Conference on Machine Learning (ICML)*, 2024.

[41] OpenAI. Hello GPT-4o, 2024. URL https://openai.com/index/hello-gpt-4o/. Accessed: October 12, 2024.

[42] OpenAI. Introducing OpenAI o1-preview, 2024. URL https://openai.com/index/introducing-openai-o1-preview/. Accessed: October 12, 2024.

[43] Emanuel Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962. ISSN 00034851. URL http://www.jstor.org/stable/2237880.

[44] 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 Jie Tan, Marc Toussaint, and Kourosh Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 23–72. PMLR, 06–09 Nov 2023.

[45] Allen Z. Ren, Jaden Clark, Anushri Dixit, Masha Itkina, Anirudha Majumdar, and Dorsa Sadigh. Explore until Confident: Efficient Exploration for Embodied Question Answering. In *arXiv preprint arXiv:2403.15941*, 2024.

[46] Pierre Sermanet, Tianli Ding, Jeffrey Zhao, Fei Xia, Debidatta Dwibedi, Keerthana Gopalakrishnan, Christine Chan, Gabriel Dulac-Arnold, Sharath Maddineni, Nikhil J Joshi, Pete Florence, Wei Han, Robert Baruch, Yao Lu, Suvir Mirchandani, Peng Xu, Pannag Sanketi, Karol Hausman, Izhak Shafran, Brian Ichter, and Yuan Cao. RoboVQA: Multimodal Long-Horizon Reasoning for Robotics. In *arXiv preprint arXiv:2311.00899*, 2023.

[47] Naman Shah, Deepak Kala Vasudevan, Kislay Kumar, Pranav Kamojjhala, and Siddharth Srivastava. Anytime Integrated Task and Motion Policies for Stochastic Environments. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9285–9291. IEEE, 2020.

[48] Naman Shah, Jayesh Nagpal, Pulkit Verma, and Siddharth Srivastava. From Reals to Logic and Back: Inventing Symbolic Vocabularies, Actions and Models for Planning from Raw Data. *arXiv preprint arXiv:2402.11871*, 2024.

[49] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[50] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-Actor: A Multi-Task Transformer for Robotic Manipulation. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 785–799, 14–18 Dec 2023.

[51] Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Kaelbling, and Joshua B. Tenenbaum. Predicate Invention for Bilevel Planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(10):12120–12129, Jun. 2023.

[52] Theodore Sumers, Kenneth Marino, Arun Ahuja, Rob Fergus, and Ishita Dasgupta. Distilling Internet-Scale Vision-Language Models into Embodied Agents. In *International Conference on Machine Learning (ICML)*, pages 32797–32818. PMLR, 2023.

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

[54] Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, 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, 2022. URL https://arxiv.org/abs/2106.14405.

[55] 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*, volume 36, pages 75993–76005, 2023.

[56] Pulkit Verma, Shashank Rao Marpally, and Siddharth Srivastava. Discovering User-Interpretable Capabilities of Black-Box Planning Agents. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, pages 362–372, 2022.

[57] 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. *arXiv preprint arXiv:2305.16291*, 2023.

[58] 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, 2019.

[59] Yufei Wang, Zhanyi Sun, Jesse Zhang, Zhou Xian, Erdem Biyik, David Held, and Zackory Erickson. RL-VLM-f: Reinforcement learning from vision language foundation model feedback. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 51484–51501, 21–27 Jul 2024.

[60] 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 *Forty-first International Conference on Machine Learning*, 2024.

[61] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. TidyBot: personalized robot assistance with large language models. *Autonomous Robots*, 47(8):1087–1102, November 2023.

[62] Kai Xi, Stephen Gould, and Sylvie Thiébaux. Neuro-symbolic learning of lifted action models from visual traces. In *Proc. ICAPS*, volume 34, pages 653–662, 2024.

[63] Jingkang Yang, Yuhao Dong, Shuai Liu, Bo Li, Ziyue Wang, Chencheng Jiang, Haoran Tan, Jiamu Kang, Yuanhan Zhang, Kaiyang Zhou, et al. Octopus: Embodied Vision-Language Programmer from Environmental Feedback. *arXiv preprint arXiv:2310.08588*, 2023.

[64] 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.

VIII. APPENDIX

A. Auxiliary Function for Predicate Invention

Algorithm 5 Scoring Functions for Invented Predicates

Input: Predicate P, skill ω, dataset of skill execution traces $\mathcal{D}$
Parameters: Threshold h.
function VALIDATEPRECOND($p, \omega, \mathcal{D}$)
    $t \leftarrow$ number of success executions where p is $True$
    $t\_total \leftarrow$ number of success executions
    $f \leftarrow$ number of failure executions where p is $False$
    $f\_total \leftarrow$ number of cases where P is $False$ in all executions
    if $t/t\_total > h$ **and** $f/f\_total > h$ **then**
        return *True*
    else
        return *False*
    end if
end function
function VALIDATEEFF($p, \omega, \mathcal{D}$)
    $t \leftarrow$ sum of $F_p(s') - F_p(s)$ in all success executions
    $t\_total \leftarrow$ number of success executions
    $eff \leftarrow True$ **if** $t > 0$ **else** $False$ ▷ hypothetical effect
    $f \leftarrow$ number of cases where $F_p(s') - F_p(s) \neq eff$ in all failure executions
    $f\_total \leftarrow$ number cases where $F_p(s') - F_p(s) \neq eff$ in all executions
    if $abs(t/t\_total) > h$ **and** $f/f\_total > h$ **then**
        return *True*
    else
        return *False*
    end if
end function

B. Auxilliary Functions for Task Proposing

Our Task Proposing algorithm scores every task proposed by the foundation model on its coverage (C), chainability (Ch) and consistency probability (Co). Below we outline the algorithmic approach used to assign these scores to proposed tasks

Algorithm 6 Auxilliary Functions to Generate and Ground Skill Sequences

function GENERATEGROUNDEDSEQUENCES($\overline{\mathcal{A}}, n$)
  $X \leftarrow$ “” ▷ prompt X to foundation model
  for $(\omega, precond_\omega, eff_\omega^-, eff_\omega^+)$ in $\overline{\mathcal{A}}$ **do**
    $X += \omega, precond_\omega, eff_\omega^-, eff_\omega^+$
  end for
  $X += y$ ▷ y additional information guiding FM to generate skill sequences
  $t\_batch \leftarrow []$
  for i in range$(0, n)$ **do**
    $\sigma = argmax(\Sigma_{i=0}^{k} log(P(u_i|X, u_0, .., u_{i-1})))$ ▷ Auto-regressive FM generation: k is max tokens for a given task
    t_batch.add(σ)
  end for
  $t\_batch\_grounded \leftarrow \{\}$
  for σ in t_batch **do**
    $t\_batch\_grounded[\sigma] = []$
    for ω in t **do**
      $\omega_{grounded} =$ MAXCOSINESIM(ω, Ω) ▷ ground proposed ω to executable skill $\omega_{grounded}$
      $t\_batch\_grounded[\sigma].add(\omega_{grounded})$
    end for
  end for
  return $t\_batch\_grounded$
end function

Algorithm 7 Auxilliary Functions for Determining Pareto-Front of Tasks

function PARETOOPTIMALITY($Score, \mathcal{D}, \overline{\mathcal{A}}, \mathcal{P}$)
  ▷ define the set of pareto-optimal tasks among proposed tasks
  $Par \leftarrow \emptyset$
  for $(t, cov, chain, cons, newcov)$ in $Score.items$ **do** $dominate = False$
    for $(t_p, cov_p, chain_p, cons_p, newcov_p)$ in Par **do**
      ▷ if some task in new task dominates task in pareto-set, swap out task in pareto-set
      if $cov \leq cov_p \wedge chain \geq chain_p \wedge cons \geq cons_p \wedge (cov < cov_p \vee chain > chain_p \vee suff > suff_p)$ **then**
        $Par.add(t, cov, chain, cons, newcov)$
        $Par.remove(t_p, cov_p, chain_p, cons_p, newcov_p)$
        $dominate = True$
      end if
      ▷ if some task in pareto-set already dominates new task, skip t
      if $cov \geq cov_p \wedge chain \leq chain_p \wedge cons \leq cons_p \wedge (cov > cov_p \vee chain < chain_p \vee suff < suff_p)$ **then**
        $dominate = True$
      end if
    end for
    if $!dominate$ **then**
      $Par.add(t, cov, chain, cons, newcov)$
    end if
  end for
  $id_{max} = argmax([\alpha_C s_p[0] + \alpha_{Ch} s_p[1] + \alpha_S s_p[2]]$ for s_p in $Par])$
  $task_{max} = Par.keys[id_{max}]$
  $skillseq_{max} = Par[Par.keys[id_{max}]][4]$
  $Q \leftarrow Par[Par.keys[id_{max}]][3]$
  return $task_{max}$, $skillseq_{max}$
end function

Algorithm 8 Auxilliary Functions for Scoring

function COVERAGE($\mathcal{D}, \sigma$)
compute the shannon entropy gain from executing the proposed skill sequence σ
Input: dataset of transitions $\mathcal{D}$, proposed skill sequence σ
$\mathcal{Q} \leftarrow 0$ ▷ construct a matrix $\mathcal{Q} \in \mathcal{M}_{|\Omega|\times|\Omega|}$ of skill-pair counts
for τ_i, τ_{i+1} in $\mathcal{D}$ **do** ▷ iterate all pairs of transitions in dataset $\mathcal{D}$
$\tau_i = (s_i, \omega(\mathbf{o})_i, s_i^{'})$
$\tau_{i+1} = (s_{i+1}, \omega(\mathbf{o})_{i+1}, s_{i+1}^{'})$
$\mathcal{Q}[\omega(\mathbf{o})_i][\omega(\mathbf{o})_{i+1}] += 1$
end for
$\mathcal{Q}^{'} \leftarrow \mathcal{Q}$ ▷ new skill-pair count initialized
$ent_{curr} = -1 \times (\mathcal{Q}/\sum \mathcal{Q}) \times log(\mathcal{Q}/\sum \mathcal{Q})$
▷ iterate all adjacent skill pairs in σ
for $(\omega(\mathbf{o})_i, \omega(\mathbf{o})_{i+1})$ in σ **do**
$\mathcal{Q}^{'}[\omega(\mathbf{o})_i, \omega(\mathbf{o})_{i+1}] += 1$
end for
$ent_{new} = -1 \times (\mathcal{Q}^{'}/\sum \mathcal{Q}^{'}) \times log(\mathcal{Q}^{'}/\sum \mathcal{Q}^{'})$
return $ent_{new} - ent_{curr}$, $Q^{'}$
end function
function CHAINABILITY($\overline{\mathcal{A}}, \mathcal{P}, t$)
Input: operator set $\overline{\mathcal{A}} = \{skill : (precond, eff^+, eff^-)\}$, predicate set $\mathcal{P}$, skill sequence σ
compute symbolic states and identify the percentage of skills where the skill sequence σ successfully executes
$len_{tot} \leftarrow len(t)$ ▷ store total number of skills
$len_{exec} \leftarrow 0$ ▷ store total number of executable skills
$s_{curr} = \{\ p : \mathcal{F}_p(s)$ for p in $\mathcal{P}\}$ ▷ track the current state
$seq = [s_{curr}]$ ▷ store sequence of abstract states
▷ iterate skills in t
for ω in t **do**
if $\forall p$ in $\overline{\mathcal{A}}[\omega].precond \wedge \mathcal{F}_p(s_{curr}) = 1$ **then**
$len_{exec} += 1$
for eff^+ in $\overline{\mathcal{A}}[\omega].eff^+$ **do** $s_{curr}[eff^+] = 1$
end for
for eff^- in $\overline{\mathcal{A}}[\omega].eff^-$ **do** $s_{curr}[eff^-] = 0$
end for
end if
▷ add current state to sequence
$seq.append(s_{curr})$
end for
return $|\frac{len_{exec}}{len_{tot}} - 0.5|$, seq
end function
function CONSISTENCY($\mathcal{T}, stateseq_t$)
Input: dataset of skill execution traces $\mathcal{D}$, grounded state sequence from executing skill sequence $stateseq_t$
$logprob \leftarrow 0$
$h \leftarrow 0.5$
for s in $stateseq_t$ **do**
$logprob += -1 \times log(\text{KDE}(s, \mathcal{T}, h))$
end for
function KDE($s, \mathcal{T}, h$)
▷ collect previous states in execution experience
$buff = \mathcal{T}.states$
▷ compute hamming distance between s and all states
$dist_{hamm} = \text{HAMMDIST}(lift(buff), lift(s))$
$kde = \exp -1 \times dist_{hamm}/h$
$kde = \frac{\sum kde}{len(kde)}$
return kde
end function
return $logprob$
end function

C. *Prompts used for* SKILLWRAPPER

Below we detail the prompts we use for predicate evaluation, skill sequence proposing, and predicate invention. The arguments within square brackets [###] are modified for dynamic-prompt construction.

```
A robot is executing a skill ‘[SKILL]‘. Given the following egocentric observation from the
robot, what is the truth value of the predicate ‘[PRED]‘? Answer with reasoning and True or
False in a separate line. Note that the blue sphere on the gripper is a part of the gripper
and is not an object, and it’s in a simulated environment so you can only tell that the
object is grasped if it’s lifted from the original surface by the gripper. Also, do not
assume the object is in the scene.
‘[PRED]‘: [SEMANTIC]
```

Fig. 6: Prompt used when leveraging foundation model as a predicate evaluator

```
Propose a set of tasks for a robot to execute along with a sequence of skills to achieve these
tasks. The robot is attempting to learn the preconditions and effects for a finite set of
skills. The robot can navigate the environment freely but only has one gripper. The robot has
access to the following skills with their associated arguments, precondition estimate and
effect estimate:

[SKILL_NAME_PRECOND_EFFECTS_DICTIONARY]

The list of objects the robot has previously encountered in the environment are:
[OBJECTS_IN_SCENE]
Book, Vase, and Bowl are on the DiningTable, and RemoteControl is on the sofa. Robot is at the
DiningTable initially.

The pairs of consecutive skills (skill1, skill2) that have been least explored are:
[LEAST_EXPLORED_SKILL_PAIRS].

You should keep in mind the type of arguments that each skill can take. Using the list of
objects and the skill preconditions / effects learned, generate 5 tasks and their sequence of
skills such that: (1) the tasks purposefully violate skill preconditions often (2) the
ordering of skills in each task is unique (3) at least 1 unexplored skill pair is used in each
task (4) all tasks have at least 8 skills in sequence.

Output only the task name and the sequence of skills to execute. Output 1 skill every new
line, following the format below:
   Task 1: Pick up the apple:
      walk_to(CounterTop)
      pick_up(Apple)

   Task 1:
```

Fig. 7: Prompt used for the skill sequence proposal

```
A robot has been programmed with the skill ‘[SKILL]‘, and it attempted to execute the skill
twice. The symbolic representations of both executions are the same while the results are
different (one succeeds and the other fails), which indicates the existing predicate set is
not sufficient to describe the precondition of the skill.

Your task is to propose a high-level and generalized predicate and the semantic meaning of the
predicate in one sentence. The predicates should be parameterized (’robot’ is not a parameter,
and empty parenthesis is allowed), and it should describe important aspects of the skill, such
as the spatial relationship between different arguments of the skill, and physical constraints
(whether the robot has empty hands, whether it’s close enough to or far away from the target),
but not any low-level predicates that involve path clearance, obstruction detection, nor
obstacle checking to check accessibility or reachability, since we don’t have measure of
those, nor any self-referring definition. For example, ‘isOpenable(obj)‘ for ‘Open(obj)‘,
‘canToggleOn(obj)‘ for ‘ToggleOn(obj)‘ are considered self-referring.

Example format:‘example_predicate‘: example_semantic_meaning.

Current predicate set: ‘[PRED_DICT]‘

(it means no predicate has been proposed if the symbolic states are empty)

You should only use the parameters of the skills if the new predicate is parametrized. Also,
you should avoid the predicates that have been tried before if this list is not empty:
‘[TRIED_PRED]‘, and you shouldn’t talk about skill executions when defining the semantic of
predicates since predicate is a description of the world state.

New predicate regarding the precondition of the skill ‘[SKILL]‘:
```

Fig. 8: Prompt used for predicate invention for precondition mismatch

```
A robot has been programmed with the skill ‘[SKILL]‘, and it attempted to execute the skill
twice. The symbolic representations of both executions are the same while the results are
different (one succeeds and the other fails), which indicates the existing predicate set is
not sufficient to describe the effect of the skill.
Your task is to propose a high-level and generalized predicate and the semantic meaning of the
predicate in one sentence. The predicates should be parameterized (’robot’ is not a parameter,
and empty parenthesis is allowed), and it should describe important aspects of the skill, such
as the change of the skill induced to the environment after execution, but not any low-level
predicates that involve path clearance, obstruction detection, nor obstacle checking to check
accessibility and reachability, since we don’t have measure of those, nor any self-referring
definition. For example, ‘isOpenable(obj)‘ for ‘Open(obj)‘, ‘canToggleOn(obj)‘ for
‘ToggleOn(obj)‘ are considered self-referring.
Example format:
‘example_predicate‘: example_semantic_meaning.
Current predicate set:
‘[PRED_DICT]‘
(it means no predicate has been proposed if the symbolic states are empty)
You should only use the parameters of the skills if the new predicate is parametrized. Also,
you should avoid similar predicates that are already in the current predicate set or have been
tried before if this list is not empty: ‘[TRIED_PRED]‘, and you shouldn’t talk about skill
executions when defining the semantic of predicates since predicate is a description of the
world state.

New predicate regarding the effect of the skill ‘[SKILL]‘:
```

Fig. 9: Prompt used for predicate invention for effect mismatch

D. PDDL Domain from Learned Operators

```
(define (domain exp˙49)
    (:requirements :strips :typing)
    (:types location object)
    (:predicates
        (hasemptyhand) (isgrasped ?o - object) (isunoccupied ?l - location)
        (isat ?o - object ?l - location)
        (isclose ?l - location)
        (isloose ?o - object)
    )
    (:action DropAt˙1
        :parameters (?l - location ?o - object)
        :precondition (and (not (hasemptyhand )) (not (isat ?o ?l))
            (isclose ?l) (isgrasped ?o) (isloose ?o) (not (isunoccupied ?l)))
        :effect (and (hasemptyhand ) (isat ?o ?l) (not (isgrasped ?o)))
    )
    (:action DropAt˙2
        :parameters (?l - location ?o - object)
        :precondition (and (hasemptyhand ) (not (isat ?o ?l))
            (isclose ?l) (isgrasped ?o) (isloose ?o) (not (isunoccupied ?l)))
        :effect (and (isat ?o ?l) (not (isgrasped ?o)))
    )
    (:action GoTo˙1
        :parameters (?l2 - location ?l1 - location)
        :precondition (and (hasemptyhand ) (isclose ?l1) (not (isclose ?l2)))
        :effect (and (not (isclose ?l1)) (isclose ?l2))
    )
    (:action GoTo˙2
        :parameters (?l2 - location ?l1 - location)
        :precondition (and (not (hasemptyhand )) (isclose ?l1) (not (isclose ?l2)))
        :effect (and (hasemptyhand ) (not (isclose ?l1)) (isclose ?l2))
    )
    (:action GoTo˙3
        :parameters (?l2 - location ?l1 - location)
        :precondition (and (hasemptyhand ) (isclose ?l1) (not (isclose ?l2)))
        :effect (and (not (hasemptyhand )) (not (isclose ?l1)) (isclose ?l2))
    )
    (:action PickUp˙1
        :parameters (?l - location ?o - object)
        :precondition (and (hasemptyhand ) (isat ?o ?l) (isclose ?l)
            (not (isgrasped ?o)) (isloose ?o) (not (isunoccupied ?l)))
        :effect (and (not (hasemptyhand )) (not (isat ?o ?l)) (isgrasped ?o))
    )
    (:action PickUp˙2
        :parameters (?l - location ?o - object)
        :precondition (and (hasemptyhand ) (not (isat ?o ?l)) (isclose ?l)
            (not (isgrasped ?o)) (not (isloose ?o)) (not (isunoccupied ?l)))
        :effect (and (not (hasemptyhand )) (isgrasped ?o) (isloose ?o))
    )
    (:action PickUp˙3
        :parameters (?l - location ?o - object)
        :precondition (and (hasemptyhand ) (isat ?o ?l) (isclose ?l)
            (not (isgrasped ?o)) (isloose ?o) (not (isunoccupied ?l)))
        :effect (and (not (isat ?o ?l)) (not (isloose ?o)) (isunoccupied ?l))
    )
    (:action PickUp˙4
        :parameters (?l - location ?o - object)
        :precondition (and (hasemptyhand ) (isat ?o ?l) (isclose ?l)
            (not (isgrasped ?o)) (isloose ?o) (not (isunoccupied ?l)))
        :effect (and (not (hasemptyhand )) (not (isat ?o ?l)) (isgrasped ?o) (isunoccupied ?l))
    )
    (:action PickUp˙5
        :parameters (?l - location ?o - object)
        :precondition (and (not (hasemptyhand )) (isat ?o ?l) (isclose ?l)
            (not (isgrasped ?o)) (isloose ?o) (not (isunoccupied ?l)))
        :effect (and (not (isat ?o ?l)) (isgrasped ?o) (isunoccupied ?l))
    )
)
```

E. PDDL Problem from Human Specification

```
(define (problem move-items)
    (:domain exp˙49)

    ;; Define the objects in the problem
    (:objects
        Vase TissueBox Bowl - object
        Sofa CoffeeTable DiningTable - location
    )

    ;; Define the initial state
    (:init
        ;; Locations of items
        (isat Vase CoffeeTable)
        (isat TissueBox Sofa)
        (isat Bowl DiningTable)

        ;; Robot's state
        (hasemptyhand)
        (isclose Sofa)

        ;; Items are initially unoccupied and loose
        (isloose Vase)
        (isloose TissueBox)
        (isloose Bowl)
        (isunoccupied CoffeeTable)
        (isunoccupied DiningTable)
    )

    ;; Define the goal state
    (:goal
        (and
            (isat Vase Sofa)
            (isat TissueBox Sofa)
            (isat Bowl Sofa)
        )
    )
)
```

F. Proposed Skill Sequences

```
Skill Sequence #1:
    [
        ”GoTo(Sofa,DiningTable)”,
        ”PickUp(Bowl,DiningTable)”,
        ”DropAt(Bowl,CoffeeTable)”,
        ”PickUp(TissueBox,Sofa)”,
        ”GoTo(CoffeeTable,DiningTable)”,
        ”DropAt(TissueBox,DiningTable)”,
        ”PickUp(Vase,CoffeeTable)”,
        ”DropAt(Vase,Sofa)”
    ]

Skill Sequence #2:
    [
        ”GoTo(Sofa,CoffeeTable)”,
        ”PickUp(TissueBox,Sofa)”,
        GoTo(CoffeeTable,DiningTable)”,
        ”DropAt(TissueBox,DiningTable)”,
        ”PickUp(Bowl,DiningTable)”,
        ”GoTo(DiningTable,CoffeeTable)”,
        ”DropAt(Bowl,CoffeeTable)”,
        ”PickUp(Vase,CoffeeTable)”
    ]

Skill Sequence #3:
    [
        ”GoTo(Sofa,DiningTable)”,
        ”PickUp(Bowl,DiningTable)”,
        ”DropAt(Bowl,DiningTable)”,
        ”DropAt(Bowl,CoffeeTable)”,
        ”GoTo(DiningTable,CoffeeTable)”,
        ”PickUp(Vase,CoffeeTable)”,
        ”GoTo(CoffeeTable,Sofa)”,
        ”DropAt(Vase,Sofa)”
    ]

Skill Sequence #4:
    [
        ”GoTo(Sofa,DiningTable)”,
        ”GoTo(DiningTable,Sofa)”,
        ”PickUp(TissueBox,Sofa)”,
        ”DropAt(TissueBox,DiningTable)”,
        ”PickUp(Vase,CoffeeTable)”,
        ”DropAt(Vase,Sofa)”
    ]

Skill Sequence #5:
    [
        ”GoTo(Sofa,DiningTable)”,
        ”PickUp(TissueBox,Sofa)”,
        ”GoTo(DiningTable,CoffeeTable)”,
        ”DropAt(TissueBox,CoffeeTable)”,
        ”PickUp(Vase,CoffeeTable)”,
        ”GoTo(CoffeeTable,Sofa)”,
        ”DropAt(Vase,DiningTable)”
    ]
```

G. Full Related Work

a) Skill Abstraction: There has been a long track of works focusing on building hierarchies that abstract away high-dimensional details with low-dimensional abstractions for planning [28, 29, 48], and those applied to robotics are usually connected to task and motion planning (TAMP) [47, 14]. These approaches however are incapable of handling high-dimensional sensory-motor signals (such as images) as input. Research on action model learning [62, 26] 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. [56] 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 model [64] and STRIPS task planner with action primitives [15, 54].

b) Predicates 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, the attempts to build classifiers for predicates from raw image inputs originated from neuro-symbolic domain [25, 36], and their initial application for robotics took a similar supervised learning approach with labeled demonstrations [38] or generated tasks [30]. After the emergence of foundation models, recent works proceed to guide skill learning with predicates generated by LLM or together with human interaction. Li et al. [32] 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 and Silver [31] and Han et al. [18] leverage human experts to provide feedback to the LLM to help it improve the learned predicates and skills.

c) Task Generation for Robotics: Automatic task proposing has been studied for active learning and curriculum learning in grid worlds and games [58, 24] to robotic domains [11, 12]. [30] 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 the foundation models, recent works have applied the idea of automatic task proposing and self-playing for exploration [40, 45], data collection [60, 63, 2], boosting skills learning [16, 57], and scene understanding [23]. 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 serve the idea of both data collection and exploration.

d) Embodied Reasoning with Foundation Models: There have been a track of work on leveraging large language models (LLMs) for embodied decision-making [21] and reasoning [22], 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 [55]. Common ways of addressing this issue include fine-tuning on datasets from a specific domains [20, 39, 5] or knowledge distillation [52, 63]. Meanwhile, many works manage to leverage pre-existing models without further training from direct visual observation [13, 40] to complete robotics tasks [23]. In all these works, the embodied reasoning ability of the foundation models serves as the central part of the systems. However, most bench-marking works evaluate the embodied reasoning ability of the models in a question-answering fashion [46, 35, 7, 6], where it remains unclear if they are capable for robotic tasks.