Language Guided Operator Learning for Goal Inference

Zachary S. Siegel
Princeton University
zss@princeton.edu

Jiayuan Mao
MIT
jiayuanm@mit.edu

Nishanth Kumar
MIT
njk@mit.edu

Tianmin Shu
Johns Hopkins University
tianmin.shu@jhu.edu

Jacob Andreas
MIT
jda@mit.edu

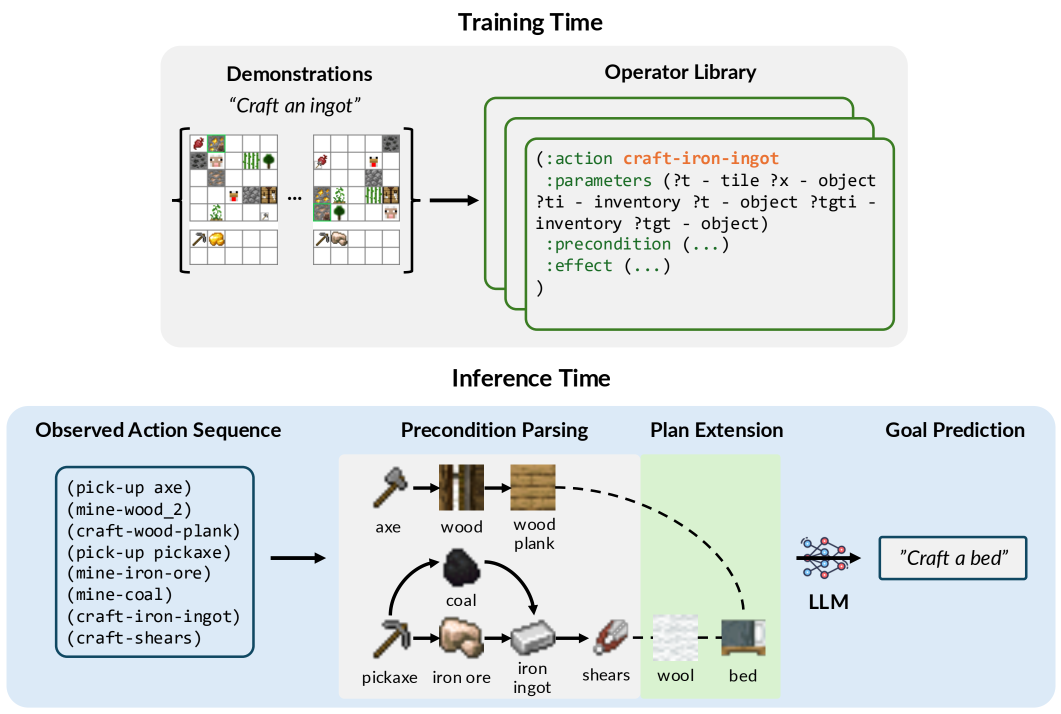
Abstract: Accurately predicting the goals of other agents is an essential skill for intelligent agents deployed in interactive environments. However, existing methods for goal inference struggle in scenarios with open-ended goal spaces or with complex action sequences. In this work, we present a novel approach for online goal inference based on library learning and explanation-based inference, guided by large language models. Our algorithm learns a library of operators from demonstrations using symbolic learning methods and guided by large language models. At inference, it predicts other agents' goals from partial plans by explaining the agents' past actions based on the learned library of operators. Specifically, we introduce an algorithm called precondition parsing which analyzes observed action sequences and hypothesizes future actions based on the relationship between observed actions. We evaluate our approach in a 2D Minecraft-like domain and show that both the library learning and explanation-based prediction significantly improve the ability to predict the goals of other agents.

Keywords: Goal Inference, Operator Learning, Language Models

1 Introduction

Effectively and efficiently predicting the goals of others is an important skill in embodied and multi-agent environments, whether one is helping a friend cook in their kitchen, reading a book, or even crossing the street. Intelligent machines that can help humans will need to be able to predict the goals of those around them. Existing work on this topic can be roughly decomposed into two categories. Direct learning or prompting-based methods based on large language models (LLMs) leverage labeled demonstrations as training data or prompts during inference [1]. However, these methods struggle when the action sequence becomes long or significantly differs from what the system has been trained on. On the other hand, "Bayesian" methods operate based on the idea of "analysis by synthesis" [2, 3] and therefore generalize better to novel goals. These methods, however, often only work in limited goal spaces and require a known transition model of the environment.

We propose a new solution that brings the advantages of both approaches. Illustrated in Figure 1, at training time, we leverage large language models (LLMs) along with symbolic operator learning techniques to learn a transition model of the environment. At inference time, learning a library of operators helps explain agents' past actions and predict what future actions might be taken, which in turn helps the overall goal prediction. Based on these learned operators, we present an algorithm called precondition parsing, which analyzes the observed sequence of actions by the inter-dependencies of actions to hypothesize future actions. The system finally predicts agent goals integrating both the commonsense priors from an LLM and the space of hypothesized future actions.


```
(:action craft-iron-ingot
 :parameters (?t - tile ?x - object
?ti - inventory ?t - object ?tgti -
inventory ?tgt - object)
 :precondition (...)
 :effect (...)
)
```

```
(pick-up axe)
(mine-wood_2)
(craft-wood-plank)
(pick-up pickaxe)
(mine-iron-ore)
(mine-coal)
(craft-iron-ingot)
(craft-shears)
```

Figure 1: **Overview.** During training time, we learn a library of operators from a dataset of task demonstrations both symbolically and guided by language. During inference time, we leverage this library of operators to perform *precondition parsing* over the observed action sequence, which outputs which actions are preconditions for other actions within the plan. Using the graph outputted by precondition parsing, we extend nodes with out-degree 0 to generate plausible extensions of the plan. Finally, we prompt a large language model to predict the goal given the plan extensions.

The contributions of this paper are two-fold. First, we propose a new operator learning algorithm that combines symbolic approaches to operator learning with LLMs to help discover additional preconditions. Secondly, we posit a new algorithm called precondition parsing, which allows efficiently predicting goals in an open-ended environment. We evaluate our approach in a 2D Minecraft-like domain and show that both the library learning and explanation-based prediction significantly improve the ability to predict the goals of other agents.

2 Related Work

Operator Learning. There is a long history of work that studies the problem of learning symbolic operators for both classical and hierarchical planning approaches [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]. We take significant inspiration from these approaches, but differ from them in two significant ways. Firstly, prior work focuses on operator learning to enable efficient planning, whereas we are interested in using operators towards goal inference. Secondly, our approach involves augmenting symbolic learning with LLMs: we are thus able to leverage semantic information (the names of provided predicates and low-level skills/actions) that many prior techniques cannot. This enables our approach to learn from smaller amounts of data, and even partial plan traces (i.e., demonstrations), where previous approaches would likely struggle. While there has been a body of work on leveraging LLMs for operator learning [16, 17, 18, 19, 20, 21], our approach integrates symbolic learning with a language-guided generation process and resamples the operators if they are not consistent with the demonstrations.

Goal Inference. Many previous works have studied the problem of building AI systems that can predict the goals of other agents [2, 3, 22, 23, 24, 25, 26]. However, these systems are not typically designed to work in open-ended settings [2, 3, 22], or have domains with a limited number of operators and predicates [23, 24, 25]. Our approach differs from these in a few ways. First, we do not assume knowledge of the operators in the domain, instead learning through demonstrations, which allows us to more easily predict goals in domains with many complex operators. Second, our approach is built to work in open-ended goal settings and does not require enumerating over all possible goal states. By leveraging our learned library of operators, we can more efficiently predict what future actions might be, which in turn helps predict the goal. One paper [27] uses a probabilistic context free grammars to predict future actions and activities from videos. Our approach, in contrast, uses the learned operators to make predictions, which could generalize towards unseen goals.

3 Problem Setting

We consider the problem of predicting the goal of an agent given a (possibly prefix) sequence of their actions. We assume all agents behavior in a deterministic environment[1], which we write as the tuple $Env = \langle \mathcal{X}, \mathcal{U}, \Phi \rangle$, where $\mathcal{X}$ is the state space, $\mathcal{U}$ is the low-level action space, and Φ is a deterministic transition function that maps $\Phi : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$. The state space $\mathcal{X}$ is defined in terms of predicates $p \in \mathcal{P}$ defined over objects in the environment. For example, the state of a kitchen environment could be represented as $\{open(\text{cabinet}), on(\text{pan}, \text{stove}), cracked(\text{egg})\}$.

Each task $T \in \mathcal{T}$, instantiated in the environment, is described by the tuple $\langle x_0, g_t \rangle$, where x_0 is the initial state of the environment and $g_t = \{p_1, p_2, ..., p_k\}$ is an (unknown) set of goal predicates. At training time, the agent receives a dataset of tasks associated with demonstrations $\langle \bar{s}, \bar{u}, T \rangle$, where $\bar{s}$ is a sequence of states and $\bar{u} = \{u_1, u_2, ..., u_H\}$ is a sequence of low-level actions taken by the agent. Our goal is to build a system that can predict g_t given $\bar{u}$ (see Figure 1). Our approach learns a library of operators $\mathcal{A}$ to help predict goals. Each operator $a \in \mathcal{A}$ is a tuple of $\langle N, \bar{v}, Pr, E^+, E^- \rangle$. N is the name, and Pr, E^+, E^- are the preconditions, add effects, and delete effects, respectively, each of which are a set of lifted atoms over the predicates P and arguments $\bar{v}$.

4 Operator Learning

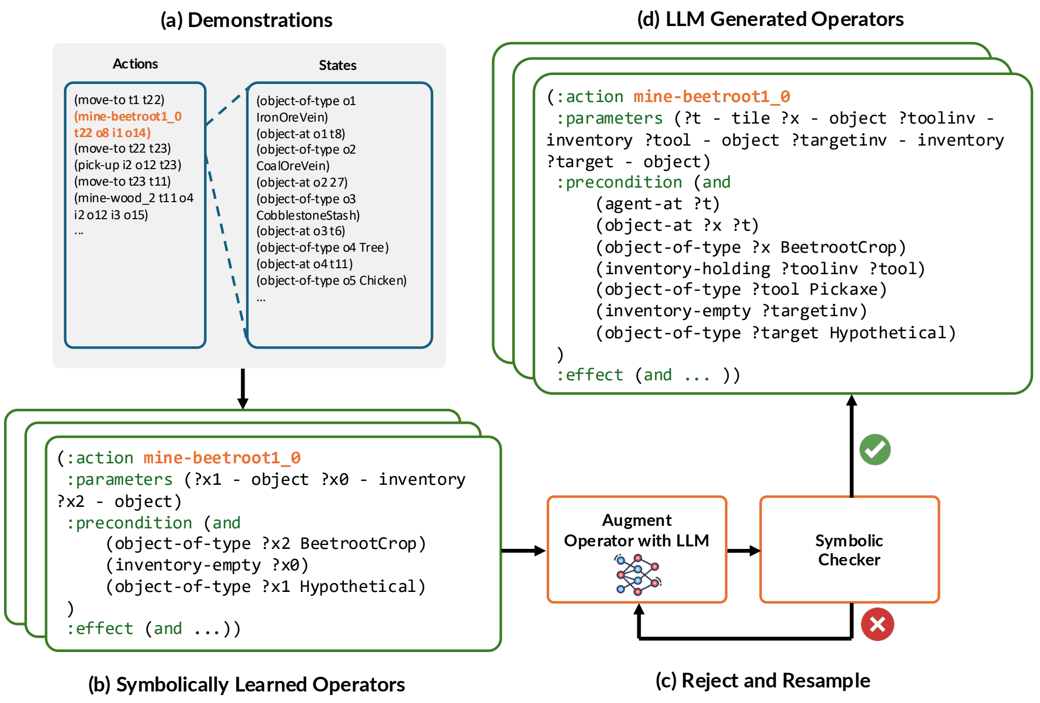
Our approach leverages a learned library of operators $\mathcal{A}$ to help predict the goals of agents (see Figure 2). We combine a symbolic approach which learns the preconditions and effects from demonstrations with a language guided approach that uses the semantics of the symbolically learned operators to infer additional preconditions.

4.1 Symbolic Operator Learning

Our operator learning algorithm is an extension of Silver et al. [12]. Here we describe the basics of their variable-lifting based algorithm. The goal is to learn a set of operators A from a dataset of demonstrations $D = \{\langle \bar{s}, \bar{u}, T \rangle\}$. We assume that each low-level action $u \in \bar{u}$ corresponds to the application of an operator $a \in \mathcal{A}$ and this mapping is a bijection (i.e. we do not learn operators that group together multiple low-level actions). Each operator (and the corresponding low-level action) is named with natural language term u. This allows us to extract the corresponding state and next state pairs associated with each operator. Let $\mathcal{O}^e$ be the affected object set of u, which is the set of all objects that are in its add effects e^+ or delete effects e^-. Since each demonstration of u involves different grounded objects, we *lift* all objects in u's affected object set by replacing all objects in u with variables of the same object type.

We learn one operator from each lifted dataset Φ_i. Let $\bar{v}$ denote the variables from lifting the dataset. To compute the add and delete effects, we simply replace the low-level action's effects $\langle e^+, e^- \rangle$ with the lifted variables $\bar{v}$ to get the *lifted* effects $\langle E^+, E^- \rangle$. To compute the preconditions, we take the

[1]We adhere to the environment and task representations outlined in Ada [16]. For detailed definitions of the state and action spaces, as well as transition models, we recommend referring to the original paper.


(a) Demonstrations
Actions
States
(d) LLM Generated Operators
Augment Operator with LLM
Symbolic Checker
(b) Symbolically Learned Operators
(c) Reject and Resample

```
(move-to t1 t22)
(mine-beetroot1_0 t22 o8 i1 o14)
(move-to t22 t23)
(pick-up i2 o12 t23)
(move-to t23 t11)
(mine-wood_2 t11 o4 i2 o12 i3 o15)
...
```

```
(object-of-type o1 IronOreVein)
(object-at o1 t8)
(object-of-type o2 CoalOreVein)
(object-at o2 27)
(object-of-type o3 CobblestoneStash)
(object-at o3 t6)
(object-of-type o4 Tree)
(object-at o4 t11)
(object-of-type o5 Chicken)
...
```

```
(:action mine-beetroot1_0
 :parameters (?x1 - object ?x0 - inventory
?x2 - object)
 :precondition (and
     (object-of-type ?x2 BeetrootCrop)
     (inventory-empty ?x0)
     (object-of-type ?x1 Hypothetical)
 )
 :effect (and ...))
```

```
(:action mine-beetroot1_0
 :parameters (?t - tile ?x - object ?toolinv -
inventory ?tool - object ?targetinv - inventory
?target - object)
 :precondition (and
     (agent-at ?t)
     (object-at ?x ?t)
     (object-of-type ?x BeetrootCrop)
     (inventory-holding ?toolinv ?tool)
     (object-of-type ?tool Pickaxe)
     (inventory-empty ?targetinv)
     (object-of-type ?target Hypothetical)
 )
 :effect (and ... ))
```

Figure 2: **Operator Learning.** **(a)** From a dataset of demonstrations, we symbolically learn the preconditions and effects of all operators corresponding to observed actions in the dataset. **(b)** Since the symbolically learned operators only include preconditions whose arguments are also in effect predicates, we use an LLM to augment the operator definitions and make the operators human interpretable. **(c)** We resample the LLM-generated operators if the proposed preconditions are not satisfied in the demonstration dataset.

intersection of all lifted atoms in the initial state before executing u. That is, the preconditions from Φ_i are $P = \cap_{u_j \in \Phi_i} s_u$, where $s_u = \text{abstract}(x_{j-1})$ when $u = u_j$.

However, this approach of computing the preconditions has one major limitation: the affected objects O^e may not include all relevant objects for the preconditions of an operator. If an operator has a precondition that is not included in the affected objects set, that predicate would not be included in the lifted state s_u. For example, suppose the operator `mine-iron-ore` has the preconditions `(holding(?t))` and `(object-of-type(?t, Pickaxe))`. If `?t` is not found in the effects, then it would not be included in $\mathcal{O}^E$. Consequently, predicates involving `?t` would not be in s_u, so the precondition `(object-of-type(?t, Pickaxe))` would not be learned. To overcome this limitation, we use LLMs to generate additional preconditions involving variables not found in $\mathcal{O}^E$.

4.2 Language Guided Precondition Learning

Using LLMs, we take a symbolically learned operator $\omega = \langle N, \bar{v}, Pr, E^+, E^- \rangle$ and propose a new operator $\omega' = \text{LLM}(\omega, \bar{\omega}, Env)$, where $\bar{\omega}$ is a small set of ground-truth operators used to few-shot prompt the LLM.

The LLM is prompted to generate a new operator $\omega' = \langle N', \bar{v}', Pr' E^{+'}, E^{-'} \rangle$ such that $E^{+'} = E^+$, $E^{-'} = E^-$, and $P' = P \cup Q$ where Q are new predicates that weren't symbolically learned. In particular, the LLM leverages its knowledge of the operator name and predicate types to propose semantically informed preconditions. The LLM also makes the variables $\bar{v}'$ human interpretable. The symbolically learned variables (e.g. `?x1, ?x2, ?x3`) are converted into human-readable variables (e.g. `?targetObject`, `?targetInventory`, `?tool`) based on the semantics of the predicates of ω.

Algorithm 1 Precondition Parsing

```
Require: Sequence of actions $\bar{u} = [u_1, u_2, \ldots, u_N]$
Ensure: Precondition graph $G = (V, E)$
 1: Initialize $V \leftarrow \bar{u}$, $E \leftarrow \emptyset$
 2: for $i = 2$ to $N$ do                                      ▷ Iterate over actions
 3:     for $j = i - 1$ down to $1$ do          ▷ Iterate over all previous actions before $u_i$
 4:         for all predicates $p \in$ Preconds$(u_i)$ do
 5:             if $p \in$ Effects$(u_j)$ then                 ▷ Check if $p$ is an effect of $u_j$
 6:                 Add edge $(u_j, u_i)$ to $E$, labeled with $p$   ▷ Indicate precond. between $u_j$ and $u_i$
 7:                 Remove $p$ from Preconds$(u_i)$       ▷ Prevent double-counting for later operators
 8:             end if
 9:         end for
10:     end for
11: end for
12: return $G = (V, E)$
```

4.3 Operator Resampling

Although the preconditions Pr' capture predicates with arguments not found in $\mathcal{O}^E$, the LLM might propose invalid preconditions that are not always satisfied. To mitigate this, we resample the operators if the proposed preconditions Pr' are not satisfied for each $u \in \bar{u}$. The preconditions for action u_i are considered satisfied if there exists a possible grounding of the precondition expression on the state s_{i-1}. If the preconditions are not satisfied, the LLM regenerates w' and repeats this process up to k iterations.

5 Goal Inference

In this section, we address predicting the goal g_t from a sequence of low-level actions $\bar{u}$ (see Figure 1). At a high level, we first use our learned library of operators to infer the relationship between actions in the observed plan using a process called precondition parsing. This step essentially explains the agent's past actions. Subsequently, we predict what future plausible steps of the plan might be given actions of the plan that haven't been used to set up other actions.

5.1 Precondition Parsing

Using the learned operator library $\mathcal{A}$, we construct a directed acyclic graph (DAG), $G = (V, E)$, using a method we call *precondition parsing* (see Algorithm 1). Each node $u_i \in V$ represents an action $u_i \in \bar{u}$, and each directed edge $(u_i, u_j) \in E$ means action u_i is a precondition for action u_j. Each edge is labeled to indicate which effect predicate of action u_i is the precondition for action u_j. Because multiple effect predicates of u_i might be preconditions for u_j, there could potentially be multiple edges between the same two nodes.

This framework allows us to reason about the underlying causality of the plan. For instance, consider an edge $e_1 = (u_1, u_2)$ where $u_1 := \texttt{open}(\texttt{cabinet}_1)$ and $u_2 := \texttt{pick-up}(\texttt{pot}_2)$, and e_1 is labeled with the predicate $\texttt{is-open}(\texttt{cabinet}_1)$. This indicates that the action $\texttt{open}(\texttt{cabinet}_1)$ must be performed before $\texttt{pick-up}(\texttt{pot}_2)$, as the effect $\texttt{is-open}(\texttt{cabinet}_1)$ of u_1 satisfies a precondition for u_2. Precondition parsing helps identify *hanging actions*, which correspond to nodes in the G with out-degree 0. Hanging actions are actions that do not satisfy the preconditions of any future step of the plan. Define $\mathcal{H} = \{u_i \in V \mid \text{out-degree}(u_i) = 0\}$ to be the set of hanging actions in G.

Assume that the plan is rational. If we have observed all steps of the plan, the hanging actions $\mathcal{H}$ should directly satisfy a predicate of the goal g_t, since hanging actions are executed not to set up some future step of the plan but for their own sake. Conversely, when the plan is only partially observed (i.e., only the first $H < N$ steps have been observed), $\mathcal{H}$ provides insight into the likely future actions, as they help to establish the preconditions necessary for a future step in the plan.

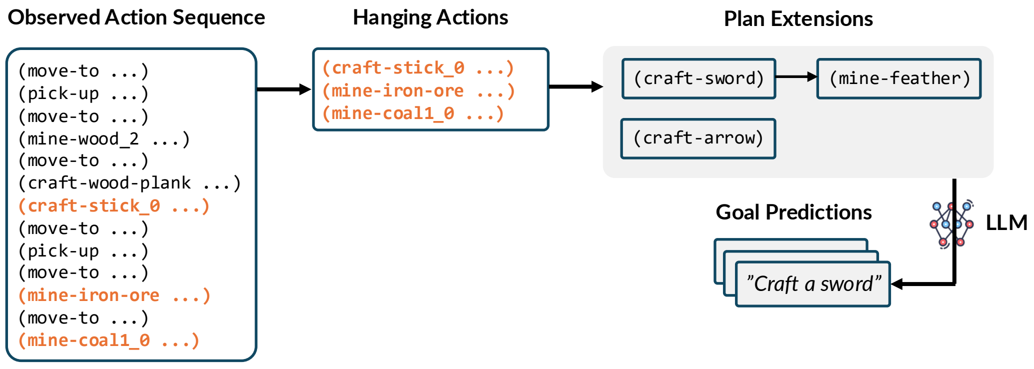


Figure 3: **Plan Extensions.** Given the action sequence of a partially observed plan, we compute the hanging actions through precondition parsing. This allows us to come up with a set of potential plan extensions, which are future actions that can be taken given the hanging actions, which we recursively expand up to K iterations. We prompt a large language model with the plan extensions to generate goal predictions.

Algorithm 2 Plan-Extend

```
Require: Precondition graph $G = (V, E)$, Operator library $\mathcal{A}$, Maximum steps $K$
Ensure: Set of plan extensions $\mathcal{R}$
 1: Compute $\mathcal{H} \leftarrow \{u_i \in V \mid \text{out-degree}(u_i) = 0\}$          ▷ Hanging actions in $G$
 2: Initialize $\mathcal{R}_0 \leftarrow \{[h] \mid h \in \mathcal{H}\}$
 3: for $k = 1$ to $K$ do                                                     ▷ Look $K$ steps ahead
 4:     $\mathcal{R}_k \leftarrow \emptyset$
 5:     for all plans $p \in \mathcal{R}_{k-1}$ do                                ▷ Check for plan extension
 6:         for all operators $a \in \mathcal{A}$ do
 7:             if Lifted(Effect(Last($p$))) $\cap$ Precond($a$) $\neq \emptyset$ then    ▷ Extend last step of $p$
 8:                 $\mathcal{R}_k \leftarrow \mathcal{R}_k \cup \{p \oplus [a]\}$
 9:             end if
10:         end for
11:     end for
12: end for
13: return $\{\mathcal{R}_1 \cup \mathcal{R}_2 \cup \cdots \cup \mathcal{R}_K\}$                 ▷ Return the set of all plan extensions
```

5.2 Plan Extension

For sequences of low-level actions $\bar{u}$ that are from partial plans, we can generate possible future actions the agent might take in a process called *plan extension* (see Algorithm 2). We use the set of hanging actions $\mathcal{H}$ and the operator library $\mathcal{A}$ to construct the set of hypothetical actions, denoted by $\mathcal{R}_1$, which contains all operators in $\mathcal{A}$ that have a precondition satisfied by the lifted effects of an action in $\mathcal{H}$. These hypothetical actions represent potential actions that may be executed by the agent. For instance, if $u_2 := \texttt{pick-up}(\texttt{pot}_2) \in \mathcal{H}$ results in the effect $\texttt{holding}(\texttt{pot}_2)$, any operator in $\mathcal{A}$ with the precondition $\texttt{holding(?pot)}$ will add its corresponding action to $\mathcal{R}_1$. We can then predict what future actions might be taken by executing each of the actions in $\mathcal{R}_{k-1}$ and forming $\mathcal{R}_k$ from the hanging actions in $\mathcal{R}_{k-1}$. We iterate this process up to K iterations.

5.3 Language Guided Goal Predictions

We predict the goals of fully observed plans by prompting an LLM: $g_t = \text{LLM}(Env, \bar{u}, \text{Eff}(\mathcal{H}), \bar{g}_t)$, where Env is the environment encoded in text descriptions, $\bar{u}$ is the sequence of observed actions, $\text{Eff}(\mathcal{H})$ represents the effect predicates of the hanging actions $\mathcal{H}$, and $\bar{g}_t$ is the set of few-shot goal prompts used to guide the LLM. We use LLMs rather than setting the goal to $\text{Eff}(\mathcal{H})$ directly since

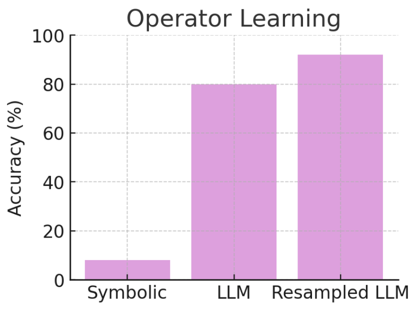


Figure 4: Accuracy of proposed operators by learning method. From left-to-right, we plot symbolic only operator learning, language guided operator learning, and language guided operator learning with resampling.

Eff($\mathcal{H}$) could contain predicates that, based on their semantics, do not make sense as a goal (e.g. $($`not`$($`inventory-holding ?ingredient1`$)$.

For partially observed plans, we predict the goal by calling $g_t = \text{LLM}(Env, \bar{u}, \mathcal{R}, \bar{g_t})$ where $\mathcal{R}$ is the set of hypothetical plan extensions, constructed recursively. The LLM helps filter by which of the candidate plan extensions $\mathcal{R}$ are most plausible given $\bar{u}$.

6 Experiments and Results

6.1 Domain

We run all experiments on Mini-Minecraft [16] a 2D procedurally generated Minecraft-like domain where agents must collect resources and craft intermediate objects to achieve their goal of obtaining certain items. There are 25 unique items that can be crafted, and plans often require several intermediate crafting steps.

6.2 Experiments

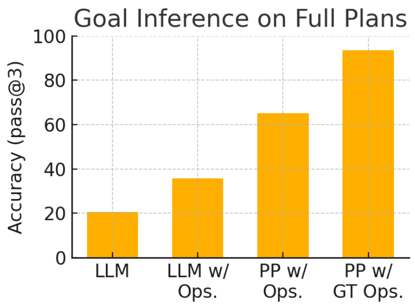
Operator Learning. We evaluate the accuracy of three different operator learning methods: (1) symbolic only, (2) language guided operator learning, and (3) language guided operator learning with resampling. We measure accuracy by comparing the proposed operators to the ground truth operators of the domain.

Goal Inference on Fully Observed Plans. We evaluate precondition parsing and LLM-only baselines for goal inference on fully observed plans. For each task instance $T = \langle \bar{u}, g_t \rangle$, we set g_t to be a random conjunction of four subgoals. Each subgoal is to hold a certain item. We generate $\bar{u}$ using the Fast Downward planner [28]. For example, one goal might be to hold shears, stick, bowl and wool. Since the plan contains intermediate steps, the challenge is figuring out which of the many crafting actions are intended as the goal.

Goal Inference on Partially Observed Plans. We evaluate precondition parsing and LLM-only baselines on fully observed plans but instead set g_t to hold one item. We generate $\bar{u}$ using Fast Downward and truncate the last three steps of the plan. Note that the partially observed setting is not necessarily strictly harder than the fully observed setting since the goals involve holding only one item.

6.3 Results and Discussion

Operator Learning. Language guided operator learning with resampling is the best method, recovering 92% of the operators in Mini-Minecraft (see Figure 4). Without resampling, the accuracy



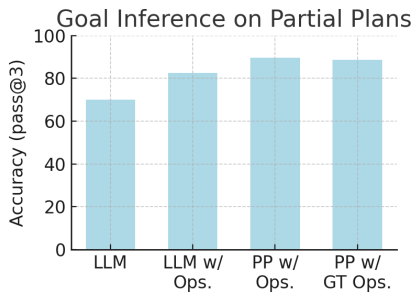


Figure 5: Pass@3 accuracy of goal inference by method. In each figure, we plot from left-to-right, (1) an LLM baseline that predicts the goal directly from $\bar{u}$, (2) an LLM baseline that predicts the goal given both $\bar{u}$ and the learned operator library $\mathcal{A}$, (3) precondition parsing with the learned operator library $\mathcal{A}$, and (4) precondition parsing with ground truth operators.

drops to 80%. The symbolic only method scores only 8%. The symbolic only method is particularly weak because many preconditions to operators in Mini-Minecraft involve variables that are not found in the effects. For example, a precondition to the operator `mine-iron-ore` is `(agent-at ?t)`. However, since `?t` is not in the effects, the variable is not lifted $\mathcal{O}^E$ or included in the preconditions. Resampling helps avoid cases where the LLM proposes incorrect preconditions. For example, when proposing preconditions for the `mine-beetroot` operator, the LLM initially (incorrectly) includes the precondition `(inventory-holding ?toolinv ?tool)`, which resampling fixes.

Goal Inference on Fully Observed Plans. Precondition parsing methods are the best at inferring the goals of partially observed plans, scoring 93.48% and 65.22% with ground-truth and learned operators $\mathcal{A}$, respectively (see Figure 5). An LLM-only baseline that predicts the goal, given actions $\bar{u}$ and learned operators $\mathcal{A}$ scores 35.87%, and an LLM-only baseline only given $\bar{u}$ scores just 20.65%, demonstrating the importance of knowing $\mathcal{A}$. Precondition parsing is especially helpful on this task since it filters out which steps of the plan are not directly satisfying one of the four subgoals but are instead setting up a future action.

Goal Inference on Partially Observed Plans. Precondition parsing methods are best, achieving 88.66% and 89.69% accuracy with ground-truth and learned operator libraries, respectively (see Figure 5). The LLM-baseline with access to $\mathcal{A}$ and $\bar{u}$ gets 82.47%, but the baseline with only $\bar{u}$ gets 70.10%. In general, all methods do relatively well on this task since even when truncating the last three steps of the plan, the semantics of $\bar{u}$ often are enough to predict the final goal. However, knowing $\mathcal{A}$ helps predict goals with unintuitive recipes.

6.4 Limitations and Future Work

Goal Inference. Future work could expand the goal space of the domain beyond the 25 included recipes to leverage precondition parsing's potential in a truly open ended setting. In addition, rather than sampling the goals directly from an LLM, one promising approach, following Bayesian inverse planning [2], is to rank the goals based on how rational the demonstration is given a goal.

Operator Learning. The resampling process during operator learning could be improved by providing more feedback to the LLM if the proposed preconditions are not satisfied, such as which specific preconditions are not satisfied. Additionally, one could relax the assumption of assuming access to the name of each operator from the dataset.

Robotics. Future work should investigate how the learned operators can perform in embodied settings. Particularly, experiments could investigate how useful the learned operators are in learning from demonstration settings for task and motion planning [29].

References

[1] T. Zhi-Xuan, P. S. Lunis, N. Fernandez Echeverri, V. Mansinghka, and J. Tenenbaum. Language Models as Informative Goal Priors in a Bayesian Theory of Mind. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 45, 2023. URL `https://escholarship.org/uc/item/31t9h5v8`.

[2] C. L. Baker, R. Saxe, and J. B. Tenenbaum. Action understanding as inverse planning. *Cognition*, 113(3):329–349, Dec. 2009. ISSN 00100277. doi:10.1016/j.cognition.2009.07.005. URL `https://linkinghub.elsevier.com/retrieve/pii/S0010027709001607`.

[3] T. Zhi-Xuan, J. L. Mann, T. Silver, J. B. Tenenbaum, and V. K. Mansinghka. Online Bayesian Goal Inference for Boundedly-Rational Planning Agents, 2020. URL `https://arxiv.org/abs/2006.07532`. Version Number: 2.

[4] G. L. Drescher. *Made-up minds: a constructivist approach to artificial intelligence*. MIT press, 1991.

[5] K. Mourao, L. S. Zettlemoyer, R. Petrick, and M. Steedman. Learning STRIPS operators from noisy and incomplete observations. *arXiv preprint arXiv:1210.4889*, 2012.

[6] H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling. Learning symbolic models of stochastic domains. *The Journal of Artificial Intelligence Research (JAIR)*, 2007.

[7] L. S. Zettlemoyer, H. M. Pasula, and L. P. Kaelbling. Learning Planning Rules in Noisy Stochastic Worlds. In *AAAI*, 2005.

[8] C. Rodrigues, P. Gérard, C. Rouveirol, and H. Soldano. Active learning of relational action models. In *International Conference on Inductive Logic Programming*, 2011.

[9] S. N. Cresswell, T. L. McCluskey, and M. M. West. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 2013.

[10] D. Aineto, S. Jiménez, and E. Onaindia. Learning STRIPS action models with classical planning. In *The International Conference on Automated Planning and Scheduling (ICAPS)*, 2018.

[11] A. Arora, H. Fiorino, D. Pellier, M. Métivier, and S. Pesty. A review of learning planning action models. *The Knowledge Engineering Review*, 2018. Publisher: Cambridge University Press.

[12] T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling. Learning Neuro-Symbolic Skills for Bilevel Planning, 2022. URL `https://arxiv.org/abs/2206.10680`. Version Number: 2.

[13] T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Pérez, L. Kaelbling, and J. B. Tenenbaum. Predicate invention for bilevel planning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2023. URL `https://ojs.aaai.org/index.php/AAAI/article/view/26429/26201`.

[14] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez. Learning Symbolic Operators for Task and Motion Planning. In *The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[15] N. Kumar, W. McClinton, R. Chitnis, T. Silver, T. Lozano-Pérez, and L. P. Kaelbling. Learning Efficient Abstract Planning Models that Choose What to Predict. In *Conference on Robot Learning (CoRL)*, 2023. URL `https://openreview.net/pdf?id=_gZLyRGGuo`.

[16] L. Wong, J. Mao, P. Sharma, Z. S. Siegel, J. Feng, N. Korneev, J. B. Tenenbaum, and J. Andreas. Learning adaptive planning representations with natural language guidance, 2023. URL `https://arxiv.org/abs/2312.08566`. Version Number: 1.

[17] Y. Chen, J. Arkin, C. Dawson, Y. Zhang, N. Roy, and C. Fan. AutoTAMP: Autoregressive Task and Motion Planning with LLMs as Translators and Checkers, Mar. 2024. URL http://arxiv.org/abs/2306.06531. arXiv:2306.06531 [cs].

[18] H. Wang, G. Gonzalez-Pumariega, Y. Sharma, and S. Choudhury. Demo2Code: From Summarizing Demonstrations to Synthesizing Code via Extended Chain-of-Thought, 2023. URL https://arxiv.org/abs/2305.16744. Version Number: 3.

[19] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf. SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Robot Task Planning, 2023. URL https://arxiv.org/abs/2307.06135. Version Number: 2.

[20] P. Smirnov, F. Joublin, A. Ceravola, and M. Gienger. Generating consistent PDDL domains with Large Language Models, 2024. URL https://arxiv.org/abs/2404.07751. Version Number: 1.

[21] Y. Ding, X. Zhang, S. Amiri, N. Cao, H. Yang, A. Kaminski, C. Esselink, and S. Zhang. Integrating Action Knowledge and LLMs for Task Planning and Situation Handling in Open Worlds. 2023. doi:10.48550/ARXIV.2305.17590. URL https://arxiv.org/abs/2305.17590. Publisher: arXiv Version Number: 2.

[22] T. Zhi-Xuan, L. Ying, V. Mansinghka, and J. B. Tenenbaum. Pragmatic Instruction Following and Goal Assistance via Cooperative Language-Guided Inverse Planning, 2024. URL https://arxiv.org/abs/2402.17930. Version Number: 1.

[23] X. Puig, T. Shu, S. Li, Z. Wang, Y.-H. Liao, J. B. Tenenbaum, S. Fidler, and A. Torralba. Watch-And-Help: A Challenge for Social Perception and Human-AI Collaboration, 2020. URL https://arxiv.org/abs/2010.09890. Version Number: 2.

[24] A. Netanyahu, T. Shu, B. Katz, A. Barbu, and J. B. Tenenbaum. PHASE: PHysically-grounded Abstract Social Events for Machine Social Perception, 2021. URL https://arxiv.org/abs/2103.01933. Version Number: 2.

[25] A. Shah, P. Kamath, J. A. Shah, and S. Li. Bayesian Inference of Temporal Task Specifications from Demonstrations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/13168e6a2e6c84b4b7de9390c0ef5ec5-Paper.pdf.

[26] F. Yan, D. Wang, and H. Hongsheng. *Robotic Understanding of Spatial Relationships Using Neural-Logic Learning*. Oct. 2020. doi:10.1109/IROS45743.2020.9340917. Pages: 8365.

[27] S. Qi, B. Jia, S. Huang, P. Wei, and S.-C. Zhu. A Generalized Earley Parser for Human Activity Parsing and Prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(8):2538–2554, Aug. 2021. ISSN 1939-3539. doi:10.1109/TPAMI.2020.2976971. URL https://ieeexplore.ieee.org/document/9018126. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[28] M. Helmert. The Fast Downward Planning System. 2011. doi:10.48550/ARXIV.1109.6051. URL https://arxiv.org/abs/1109.6051. Publisher: arXiv Version Number: 1.

[29] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4(1):265–293, 2021.