Open-World Planning via Lifted Regression with LLM-Inferred Affordances for Embodied Agents

Xiaotian Liu[1], **Ali Pesaranghader**[2], **Hanze Li**[1], **Punyaphat Sukcharoenchaikul**[1], **Jaehong Kim**[3], **Tanmana Sadhu**[2], **Hyejeong Jeon**[3], and **Scott Sanner**[1,4]

[1] University of Toronto, Toronto, Canada [2] LG Electronics, Toronto AI Lab, Toronto, Canada

[3] LG Electronics, Seoul, South Korea [4] Vector Institute of Artificial Intelligence, Toronto, Canada

{xiaotian.liu, litos.li, punyaphat.sukcharoenchaikul}@mail.utoronto.ca

{ali.pesaranghader, jaehong02.kim, tanmana.sadhu, hyejeong.jeon}@lge.com

ssanner@mie.utoronto.ca

Abstract

Open-world planning with incomplete knowledge is crucial for real-world embodied AI tasks. Despite that, existing LLM-based planners struggle with long chains of sequential reasoning, while symbolic planners face combinatorial explosion of states and actions for complex domains due to reliance on grounding. To address these deficiencies, we introduce LLM-Regress, an open-world planning approach integrating *lifted regression* with LLM-generated affordances. LLM-Regress generates sound and complete plans in a compact lifted form, avoiding exhaustive enumeration of irrelevant states and actions. Additionally, it makes efficient use of LLMs to infer goal-related objects and affordances without the need to predefine all possible objects and affordances. We conduct extensive experiments on three benchmarks and show that LLM-Regress significantly outperforms state-of-the-art LLM planners and a grounded planner using LLM-generated affordances. Our experimental results highlight the potential of LLM-Regress for sound and complete open-world planning for embodied AI tasks.

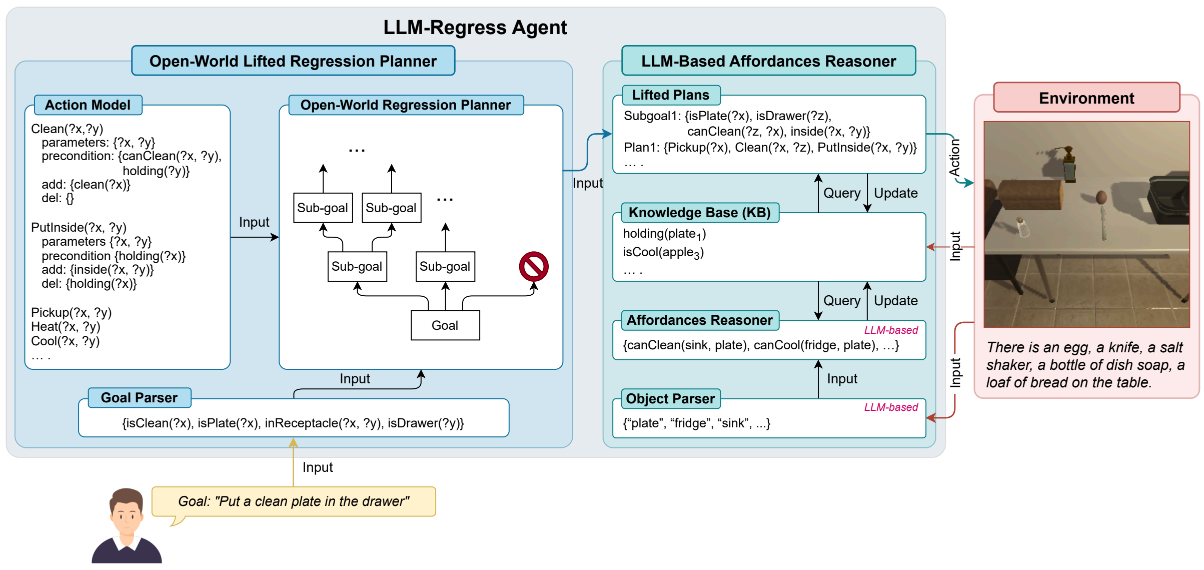
1 Introduction

Planning in open-world environments with incomplete knowledge is a necessary requirement for embodied agents to perform many real-world tasks. For instance, when tasked with "clean the kitchen", an agent must infer the possible presence of certain objects (e.g., are there dirty plates to be cleaned?), their relational information (e.g., are there plates on the dining table or the kitchen counter?), and determine appropriate action affordances (e.g., should the plate be cleaned in a sink instead of using a broom?).

Although classical planning methods provide sound and complete solutions for long-horizon planning tasks, they rely heavily on the closed-world assumption (CWA), which requires a fully specified initial state (Ghallab et al., 2004). Previous efforts to adopt classical planning methods for open-world tasks typically involve solving a "closed" sub-problem and re-planning as new information becomes available (Perera et al., 2015; Jiang et al., 2019; Hanheide et al., 2017). Despite these adaptations, existing open-world planners face two major limitations: (1) the reliance on predefined object types and action affordances, and (2) the challenge of determining which objects and actions are relevant to the goal to avoid a combinatorial explosion of reasoning about irrelevant information. These limitations make the current approaches impractical for dynamic, open-world settings, such as households, where thousands of unique objects may exist in various configurations.

In a different vein, large language models (LLMs) have demonstrated remarkable commonsense reasoning capabilities (Yao et al., 2024; Ouyang et al., 2022), a crucial aspect of open-world tasks. Unlike classical planning methods, LLMs do not require structured inputs or explicit knowledge modelling, making them an attractive alternative to solving goal-oriented open-world problems (Yao et al., 2022; Shinn et al., 2024). However, growing evidence raises concerns about the effectiveness of LLMs in long-horizon planning tasks (Valmeekam et al., 2023). LLMs are also prone to hallucination and are highly sensitive to prompt input (Huang et al., 2023), undermining their reliability for practical applications.

In this work, we propose to combine lifted regression with LLM commonsense reasoning to address *open-world* planning challenges for embodied agents. Lifted regression is sound and complete (Ghallab et al., 2004; Liu and Lakemeyer, 2010). *Critically, instead of searching from the initial state, lifted regression employs goal-directed backward search to derive relevant state descriptions of subgoals, thus filtering out irrelevant objects and actions.* Furthermore, the plans generated by lifted



LLM-Regress Agent
Open-World Lifted Regression Planner
Action Model
Clean(?x,?y)
parameters: {?x, ?y}
precondition: {canClean(?x, ?y), holding(?y)}
add: {clean(?x)}
del: {}
PutInside(?x, ?y)
parameters {?x, ?y}
precondition {holding(?x)}
add: {inside(?x, ?y)}
del: {holding(?x)}
Pickup(?x, ?y)
Heat(?x, ?y)
Cool(?x, ?y)
… .
Input
Open-World Regression Planner
…
Sub-goal
Sub-goal
…
Sub-goal
Sub-goal
Goal
Input
Goal Parser
{isClean(?x), isPlate(?x), inReceptacle(?x, ?y), isDrawer(?y)}
Input
Goal: "Put a clean plate in the drawer"
Input
LLM-Based Affordances Reasoner
Lifted Plans
Subgoal1: {isPlate(?x), isDrawer(?z), canClean(?z, ?x), inside(?x, ?y)}
Plan1: {Pickup(?x), Clean(?x, ?z), PutInside(?x, ?y)}
… .
Query
Update
Knowledge Base (KB)
holding(plate$_1$)
isCool(apple$_3$)
… .
Query
Update
Affordances Reasoner
LLM-based
{canClean(sink, plate), canCool(fridge, plate), …}
Input
Object Parser
LLM-based
{"plate", "fridge", "sink", ...}
Action
Input
Input
Environment
There is an egg, a knife, a salt shaker, a bottle of dish soap, a loaf of bread on the table.

Figure 1: **Illustrative Example for LLM-REGRESS.** The LLM-REGRESS Agent has two major components: Open-World Lifted Regression Planner and LLM-Based Affordances Reasoner. The former is responsible for regressing lifted plans and the latter for parsing objects' names as well as generating their affordances.

regression are parameterized with variables rather than specific objects, eliminating the need for predefined objects or the exhaustive enumeration of object-action combinations.

To this end, we introduce LLM-REGRESS, an open-world planning approach that leverages lifted regression to produce sound and complete plans while utilizing large language models (LLMs) to infer unseen objects and action affordances from natural language observations. An overview of LLM-REGRESS can be seen in Figure 1. We perform experiments on the ALFWorld dataset (Shridhar et al., 2020), a diverse embodied household benchmark that is partially observable with complex affordance reasoning (an example task is shown in Appendix A). Additionally, we evaluated LLM-REGRESS on two curated datasets, ALFWorld-Afford and TableTop-Afford, which introduce more complex goals and affordances.

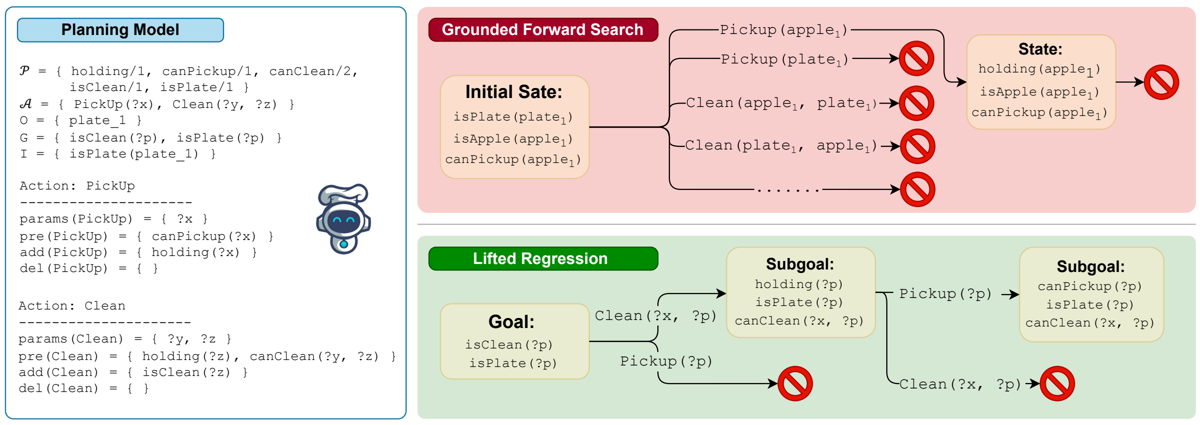
We show that LLM-REGRESS outperforms other baselines on all three benchmarks while using only a small fraction of the LLM tokens of competing methods and without relying on few-shot examples. The main contributions of this work are as follows:

- We are the first to leverage lifted regression to open-world embodied AI planning, generating sound and complete lifted plans without exhaustive state and action enumeration.
- We introduce LLM-REGRESS, an open-world planning approach that integrates lifted regression with LLMs, enabling reasoning without predefined object types or action affordances.
- LLM-REGRESS outperforms existing LLM-based and grounded planners (using LLM-generated affordances) for 3 embodied AI benchmarks in success rate, query token efficiency, and total task execution time.

2 Background and Related Work

LLM-Driven Embodied AI Planning. Many recent works have investigated LLM's planning capability for embodied agents. Works such as (Ahn et al., 2022; Valmeekam et al., 2023; Hazra et al., 2024) use LLMs as heuristics to choose the best action to fulfill the agent's goal. Other works like (Yao et al., 2022; Shinn et al., 2024; Huang et al., 2022) rely on LLMs to directly generate plans and iteratively refine them via self-reflection. Retrieval-based approaches have also been explored to retrieve past data to improve decision-making (Wang et al., 2023, 2024). Despite showing promising results, LLM-based planners still lack the interpretability and robustness required for many embodied AI problems (Valmeekam et al., 2023).

Grounded Forward Planning with LLMs. Most recent works have attempted to integrate LLMs with classical planning approaches (Arora and Kambhampati, 2023; Guan et al., 2023; Xie et al., 2023a; Hazra et al., 2024). The state-of-the-art


```
Planning Model

𝒫 = { holding/1, canPickup/1, canClean/2,
      isClean/1, isPlate/1 }
𝒜 = { PickUp(?x), Clean(?y, ?z) }
O = { plate_1 }
G = { isClean(?p), isPlate(?p) }
I = { isPlate(plate_1) }

Action: PickUp
---------------------
params(PickUp) = { ?x }
pre(PickUp) = { canPickup(?x) }
add(PickUp) = { holding(?x) }
del(PickUp) = { }

Action: Clean
---------------------
params(Clean) = { ?y, ?z }
pre(Clean) = { holding(?z), canClean(?y, ?z) }
add(Clean) = { isClean(?z) }
del(Clean) = { }
```

Figure 2: **Grounded Forward Search vs. Lifted Regression.** The results demonstrate that Grounded Forward Search (top) fails to generate feasible without a complete initial state, whereas Lifted Regression(bottom) successfully produces plan traces and subgoals, enabling open-world reasoning.

planners operate under a closed-world assumption (CWA) and rely on forward search in fully grounded task representations (Helmert, 2006; Hoffmann and Nebel, 2001; Calvanese et al., 2018). As a result, grounded forward planning methods can only generate valid plans if all relevant objects are known—an unrealistic scenario in open-world settings. When faced with partial knowledge, an agent must exhaustively enumerate actions across all objects, leading to a combinatorial explosion (as the number of grounded actions grows exponentially with the number of parameters and objects). Consequently, most existing hybrid planning methods require a fully known initial state with a limited set of predefined object types and action affordances.

Lifted Regression Planning. An alternative sound and complete approach to grounded forward search is lifted regression planning (Reiter, 2001; Ghallab et al., 2004; Sanner and Boutilier, 2009). Instead of searching forward from the initial state, it performs a backward search from the goal. Critically, *lifted* regression operates in a parameterized space where *state and action descriptions use variables* instead of specific ground objects, eliminating the need for an exhaustive enumeration of states and actions. While most lifted regression algorithms are designed for closed-world problems, their lifted representation also makes them *suitable for open-world settings*, where variables can be instantiated only when appropriate objects are observed. Intuitively, a lifted plan serves as a *plan template*, focusing on object properties rather than specific instances—e.g., instead of searching for a specific "*sink*$_3$ *to clean a plate*", it identifies objects via existentially quantified variables (e.g. $?x$) satisfying relevant affordances, such as "*some $?x$ that can clean a plate*". The integration of LLMs with lifted regression remains largely unexplored, and we demonstrate for the first time the potential of combining lifted regression with LLMs to solve open-world embodied AI problems.

3 Methodology: LLM-Regress

In this section, we provide a detailed overview of LLM-Regress and its two key components: the *Open-World Lifted Regression Planner* and the *LLM-Affordances Reasoner*. An overview of LLM-Regress is shown in Figure 1. An example of LLM-Regress solving an ALFWorld task is shown in Appendix B.

3.1 Language Interface to Lifted Planning

In this work, we focus on language-based embodied AI tasks, where both instructions and observations are represented in natural language. Following the existing model-based approaches (Song et al., 2023; Chen et al., 2024; Huang et al., 2022), we assume the agent possesses a set of high-level skills encoded as action models using STRIPS-like syntax (Fikes and Nilsson, 1971). Before the planner can search for feasible plans, the agent's goal must first be translated from natural language into a formal symbolic representation.

3.1.1 Goal Parser

We adopt the goal conversion methods from (Song et al., 2023; Xie et al., 2023b), which can reliably transform natural language instructions into planning goals by prompting LLMs with the agent's action model with few-shot examples. An example of this prompt is provided in Appendix C. As shown in Figure 1, when the agent receives the instruction:

"Put a clean plate in the drawer."

it translates this into the following structured goal representation:

$$\{\texttt{isClean}(?x), \texttt{isPlate}(?x),\\ \texttt{inReceptacle}(?x, ?y), \texttt{isDrawer}(?y)\}.$$

3.1.2 Open-World Planning Formalism

With a structured action model and goal, we can formalize open-world regression problems. We use a set-theoretic representation to define the open-world lifted regression planning problem as $\Pi = \langle \mathcal{P}, \mathcal{A}, O, G, I\rangle$, where $\mathcal{P}$ is a set of *predicate symbols*, $\mathcal{A}$ is a set of *action schemas*, O is the set of *known objects*, G is the agent's *goal*, and I represents the agent's *initial knowledge*. An atom $p_i(T_i)$ consists of a predicate symbol $p_i \in \mathcal{P}$ and an n-tuple of terms $T = \langle t_1, \ldots, t_n\rangle$, with variables in T denoted as $\text{vars}(T)$. An atom is *lifted* if $\text{vars}(T) \neq \emptyset$. Each action schema A is defined as $A = \{\mathit{params}(A), \mathit{pre}(A), \mathit{add}(A), \mathit{del}(A)\}$, where $\mathit{params}(A)$ is the set of parameters, $\mathit{pre}(A)$ is the precondition, $\mathit{add}(A)$ consists of predicates added to a subgoal, and $\mathit{del}(A)$ contains predicates removed from a subgoal. Figure 2 illustrates a formalized example of an open-world planning problem (i.e., only a subset of objects O are known in the initial state).

3.1.3 Lifted Subgoal and Plans

With a formalized planning model, we can then perform open-world lifted regression through a backward search from the goal. This process generates a set of lifted plans along with their corresponding subgoals. A lifted subgoal can be interpreted as the preconditions that must be satisfied to execute a plan. An example of subgoal, illustrated in Figure 2:

$$\{\texttt{isPickup}(?p), \texttt{isPlate}(?p), \texttt{canClean}(?x, ?p)\},$$

which states that the agent needs to "find some plate $?p$ and some $?x$ to clean the plate with" in order to excute the plan:

$$< \texttt{PickUp}(?p), \texttt{Clean}(?x, ?p) >$$

3.1.4 Lifted Regression Operations

Regression planning begins with the goal and applies *regression operations* (inverse of progression) for only *relevant* actions until the initial condition is met or a stopping criterion is reached. We define both concepts as the following:

- $\text{RELEVANT}(a, s) = (s \cap (\text{add}(a) \cup \text{del}(a)) \neq \emptyset) \wedge (s \cap \text{del}(a) = \emptyset) \wedge (s \cap \text{add}(a) = \emptyset)$ determines whether an action a contributes to achieving the subgoal s. An action is relevant if its effects add something to the subgoal without contradiction.
- $\text{REGRESS}(s, a) = (s \setminus \text{add}(a)) \cup \text{pre}(a)$ computes the preceding subgoal s before executing action a.

We also use other standard logical operations such as SUBSTITUTE, UNIFY, and STANDARDIZE, defined in Appendix K.

Referring back to Figure 2, we want to determine which action can be regressed from the goal. Notice that $\text{add}(\texttt{Clean})$ contains $\texttt{isClean}(?z)$, which can be unified with $\texttt{isClean}(?p)$ using the substitution $\theta = \{?z/?p\}$. Hence,

$$G \cap \text{add}(\texttt{Clean}) \neq \emptyset,$$

and since $\text{del}(\texttt{Clean}) = \emptyset$, the action $\texttt{Clean}$ is relevant with respect to G.

In contrast, $\text{add}(\texttt{PickUp}) = \{\texttt{holding}(?x)\}$ does not share any predicates with G, meaning

$$G \cap \text{add}(\texttt{PickUp}) = \emptyset.$$

Thus, $\texttt{PickUp}$ is not relevant for achieving G, while $\texttt{Clean}$ is.

The regressed subgoal $\text{REGRESS}(G, \texttt{Clean})$ is given by:

$$G \setminus \text{add}(\texttt{Clean}(?y, ?p)) \cup \text{pre}(\texttt{Clean}(?y, ?p))$$

$$= \{\texttt{isPlate}(?p), \texttt{holding}(?p), \texttt{canClean}(?y, ?p)\}.$$

3.1.5 Open-World Lifted Regression Algorithm

Open-world lifted-regression takes a problem specification $\Pi = \langle \mathcal{P}, \mathcal{A}, O, G, I\rangle$ and a plan length threshold τ, returning a set S of subgoals paired with plans (action sequences). In each iteration, a pair of subgoals and plan, (g, π), is dequeued from the frontier. If π reaches length τ, the pair is added to S, representing a feasible lifted plan; otherwise, for each action in $\mathcal{A}$, the action is standardized to avoid variable conflicts and unified with g. If a action is determine to be RELEVANT, the action is appended to π and used to compute a regressed

Algorithm 1 Open-World Lifted-Regression $\Pi = \langle \mathcal{P}, \mathcal{A}, O, G, I \rangle$, τ

```
1: $S \leftarrow \{\}$
2: Frontier $\leftarrow \{(G, \pi = [])\}$
3: Visited $\leftarrow \{G\}$
4: while Frontier $\neq \emptyset$ do
5:     $g, \pi \leftarrow$ POP(Frontier)
6:     if len($\pi$) = $\tau$ then
7:         $S$.add(($g, \pi$))
8:     else
9:         RegressibleActions = {}
10:        for each $A$ in $\mathcal{A}$ do
11:            $A' \leftarrow$ STANDARDIZE($A$)
12:            $\theta \leftarrow$ UNIFY($A', g$)
13:            if RELEVANT($\theta(A'), \theta(g)$) then
14:                RegressibleActions.add($A'$)
15:                $\pi$.append($A'$)
16:                $g' \leftarrow$ REGRESS($\theta(g), \theta(A')$)
17:                if $g'$ not in Visited then
18:                    Visited.add($g'$)
19:                    Frontier.add(($g', \pi$))
20:                end if
21:            end if
22:        end for
23:        if RegressibleActions = $\emptyset$ and $g' \notin$ Visited then
24:            $S$.add(($g, \pi$))
25:        end if
26:    end if
27: end while
28: return $S$
```

goal $g' = \text{REGRESS}(g, A)$. If g' is new, it is added to the *Visited* set and enqueued in the frontier. If no regressible action exists for a subgoal, (g, π) is added to S. This process continues until the frontier is empty, returning S. The details are outlined in Algorithm 1.

3.2 Lifted Planning via Regression to Affordances-driven Actions

Once the agent obtains lifted subgoals and plans via regression, it explores and interacts with the environment to discover objects that can satisfy a subgoal, enabling plan execution. Each time the agent interacts with the environment, it uses the *LLM Object Parser* to extract a set of objects from observations and save them as name strings. The agent then checks whether the non-affordance portion of any subgoals can be satisfied by agent's Knowledge Based (KB). If such a subgoal exists, the agent generates object affordances via *LLM Affordances Query* based on the list of observed object name strings. Once the agent's KB satisfies a subgoal, it instantiates the variables in the lifted subgoal and plan, and then executes the actions in the plan sequentially. We call this module *LLM Affordances Reasoner*. A detailed algorithm is shown in Algorithm 2 with an explanation in Appendix L.

3.2.1 LLM Object Parser

In domains where observations can be expressed in natural language, LLMs can directly extract objects from descriptions and store them as name strings. We assume the agent maintains a set of objects O, initialized with known objects (e.g., those mentioned in the goal) and updated dynamically as it discovers new ones during exploration and task execution. Since objects are stored as name strings, the agent does not require a predefined set of object types. For example, if the agent encounters the sentence:

"There is a table, a microwave, and an elephant in the room."

it can use an LLM to extract the object list:

$$O = \{\text{"table"}, \text{"microwave"}, \text{"elephant"}\}$$

without pre-predefining object types. The prompt used for extraction is provided in Appendix D.

3.2.2 LLM Affordances Query

LLM affordances queries rely on two key components: (1) a list of object name strings, i.e., O, obtained via the LLM Object Parser, and (2) a predefined set of affordance predicate types. When the agent finds a subgoal that can be satisfied by its current KB without affordances predicates, it then queries the LLM to determine which existing objects can satisfy the affordance portion of the subgoal. Using the example shown in Figure 2, let's assume the agent is currently holding a "plate":

$$\{\texttt{holding(plate)}, \texttt{isPlate(plate)}\} \subseteq \text{KB}$$

and observes a set of new objects:

$$O = \{\text{"table"}, \text{"apple"}, \text{"sink"}\}.$$

We want to determine whether the subgoal:

$$\text{g} = \{\texttt{isPlate}(?p), \\ \texttt{holding}(?p), \texttt{canClean}(?x, ?p)\}$$

can be satisfied. From agent's KB, we can determine that the *non-affordances portion* of g can be satisfied. We then use LLM to query whether `canClean(?x, ?p)` holds for some objects in O. For example, if the LLM returns the object "sink", we can then add `canClean(plate, sink)` to the KB, and the complete subgoal {`holding(plate)`, `isPlate(plate)`, `canClean(plate, sink)`} can be used to initiate a plan. The prompt for this query is shown in Appendix E.

Algorithm 2 LLM-Regress Reasoner

```
Require: LiftedPlans, $O$, KB, AffordPredTypes
 1: FailedAff $\leftarrow \{\}$
 2: while $G$ is not satisfied in KB do
 3:     for all $(g, \pi) \in$ LiftedPlans do
 4:         $g_a \leftarrow \{p \in g \mid \text{type}(p) \in \text{AffordPredTypes}\}$
 5:         $g_{na} \leftarrow g \setminus g_a$
 6:         if KB satisfies $g_{na}$ then
 7:             $F_{na} = \text{KB}.qeury(g_{na})$
 8:             $F_a \leftarrow \text{LLMAffQuery}(F_{na}, \text{FailedAff}, O)$
 9:             KB.add($F_a$)
10:         end if
11:         if $g_{na} \cup g_a$ is satisfied then
12:             for all $a \in \pi$ do
13:                 $a = \text{SUBSTITE}(a, F_a \cup F_{na})$
14:                 Act($a$)
15:                 if Act($a$) fails then
16:                     FailedAff.add($F_a$)
17:                     KB.remove($F_a$)
18:                 else
19:                     PROGRESS(KB, $a$)
20:                 end if
21:             end for
22:         end if
23:     end for
24:     $O \leftarrow O \cup \text{LLMObjectParser}(\text{RandomExplore}())$
25: end while
```

4 Experiments

We evaluate LLM-REGRESS against three embodied AI benchmarks, and compare it with state-of-the-art LLM planners and a grounded forward search planner (with LLM-generated affordances). We evaluate the results with respect to success rate planning duration, and the number of tokens used in queried LLMs.

4.1 Assumptions and Design Details

We assume a static environment with deterministic actions. The agent is given a set of action schemas without predefined object types (except for objects specified in the goal) or action affordances. The action schema follows a STRIPS-style syntax, as in Appendices G and H. We use `GPT-4o` as the underlying LLM for all baselines and conducted all experiments on a computer equipped with a modern `Intel i7` processor and `32 GB` of RAM.

4.2 Benchmarks

ALFWorld. ALFWorld (Shridhar et al., 2020) is a text-based virtual household environment where an agent must perform common household tasks based on a set of natural language instructions. It includes 134 environments spanning 6 different task categories. The environment is designed to support open-world reasoning, featuring partial observability, 125 distinct object types, and approximately $93,750$ possible affordances among objects. These affordances define whether an object (individually or in pairs) can be heated, cooled, cleaned, or turned on/off. We do not provide object types, which consequently requires the agent to discover new objects dynamically during task execution. Further details on ALFWorld can be found in (Shridhar et al., 2020).

ALFWorld-Afford. AFLWorld tasks limit affordances reasoning by restricting actions (e.g., heating only via a microwave, cooling via a fridge, cleaning via a sink), which encourage memorization over affordances reasoning. To address this, we curated ALFWorld-Afford, which increases both affordances diversity and goal types. Details of these tasks can be found in Appendix I.

Tabletop-Afford. Robotic tabletop manipulation such as (Shridhar et al., 2022) is also commonly used to evaluate embodied planning performance. However, existing tabletop manipulation tasks do not consider affordances reasoning. To extend our study, we also curated a set of 30 tabletop manipulation tasks using PDDLGym (Silver and Chitnis, 2020) with a language-based observations translator. We added action affordances constraints for placing objects on top of others. For example, we allow actions such as “putting a cup on a tray” and prevent actions such as “putting a cup on an apple”. Appendix J provides the details of Tabletop-Afford.

4.3 Evaluation Metrics

We consider three metrics for our experimental evaluation: (1) **Success Rate**, the percentage of successfully completed tasks within the permitted steps (i.e., 50 steps for the ALFWorld benchmarks, and 20 for Tabletop-Afford), (2) **Number of Tokens**, the total number of tokens used in LLM queries, and (3) **Task Duration**, the time taken by the agent to complete the task, measured in seconds. All experiments are conducted 3 times and the average and standard deviation are reported.

4.4 Baselines

LLM Baselines. We compare LLM-REGRESS with the state-of-the-art LLM reasoner **ReAct** (Yao et al., 2022), which generates thoughts, tracks reasoning, and updates actions. Unlike LLM-REGRESS, ReAct relies on at least two few-shot examples per task. To evaluate the impact of examples, we also evaluate LLM-REGRESS against two ReAct variations: (1) **ReAct w/ Examples**,

Table 1: Performance Comparison across the ALFWorld, ALFWorld-Afford, and Tabletop-Afford Benchmarks.

Method	**ALFWorld**			**ALFWorld-Afford**			**Tabletop-Afford**		
	Success Rate	**Tokens**	**Duration**	**Success Rate**	**Tokens**	**Duration**	**Success Rate**	**Tokens**	**Duration**
Grounded Planner w/ LLM Afford.	0.35±0.09	4K	13 sec	0.29±0.07	4K	19 sec	0.38±0.05	1K	10 sec
Direct LLM (GPT-4o)	0.21±0.12	1K	20 sec	0.12±0.09	1.5K	23 sec	0.27±0.04	0.4K	20 sec
ReAct									
w/ Model Description	0.33±0.02	35K	34 sec	0.17±0.05	42K	39 sec	0.25±0.02	8K	35 sec
w/ Examples	0.70±0.05	40K	33 sec	0.57±0.02	56K	41 sec	0.72±0.03	12K	25 sec
LLM-Regress (Ours)	**0.95±0.02**	**0.5K**	**5 sec**	**0.84±0.03**	**0.6K**	**8 sec**	**0.83±0.02**	**0.2K**	**5 sec**

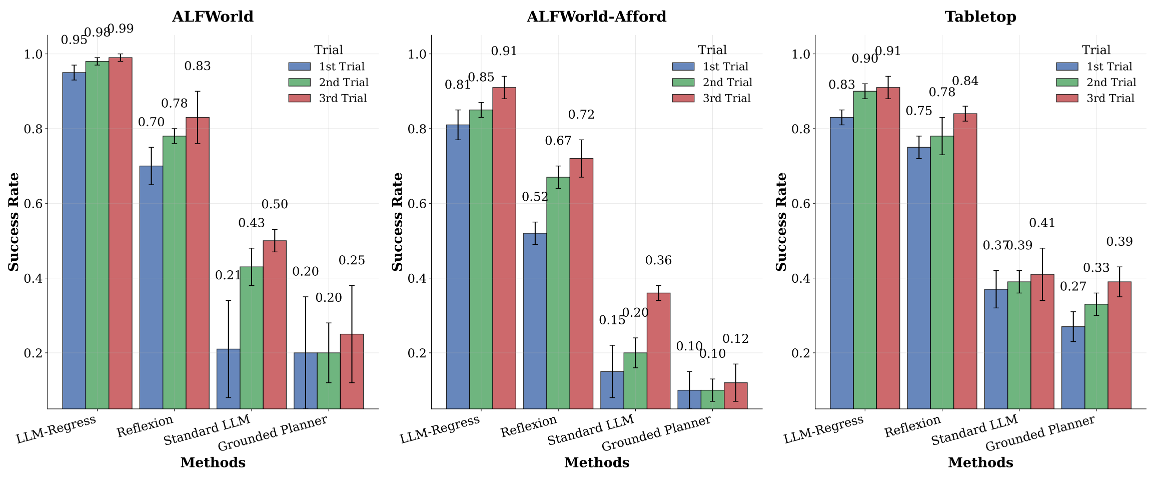


Figure 3: **Multi-trial Performance Comparison** against ALFWorld, ALFWorld-Afford, and Tabletop-Afford.

the original version using few-shot examples—and (2) **ReAct w/ Model Description**, a version where task examples are replaced with a natural language description of the **Action Model** used in LLM-Regress. We also test a baseline that ablates the reasoning and directly prompts `GPT-4o` to produce actions (**Direct LLM**).

Grounded Forward Search. We built a grounded forward search planner that uses LLM-generated affordances. We provide a planner (Helmert, 2006) with the action model, goal, and initial state (with all objects and locations) in PDDL. We then prompt LLMs to generate the necessary affordances information (see Appendix F), incorporate it into the initial state, and generate a plan using the FD planner (Helmert, 2006). We refer to this method as **Grounded Planner w/ LLM Affordances**. If a plan is generated, we then evaluate it in the simulator, else we consider it as a failure.

Multi-Trial LLM Baselines. Another notable LLM-based planner, **Reflexion** (Shinn et al., 2024), enhances ReAct by reflecting on past attempts. In contrast, LLM-REGRESS maintains a structured KB with affordances information that can be reused, akin to Reflexion's strategy. To evaluate structured KB transfer, we compare Reflexion, Direct LLM, and Grounded Planner across 3 successive trials to evaluate their respective performance improvements from experience.

5 Results and Discussion

LLM-REGRESS outperforms LLM and grounded planning baselines across all benchmarks, achieving higher success rates with fewer tokens and reduced planning time. Unlike grounded planners requiring exhaustive affordances enumeration, LLM-REGRESS generates affordances only when needed, improving efficiency and reliability. It is also more token-efficient than LLM planners while maintaining superior success rates. Though performance declines with task complexity, LLM-REGRESS remains the most effective. Its structured KB further enhances knowledge transfer, boosting performance across multiple trials. The primary results are show in Table 1 with a breakdown by each task type in Appendix M.

5.1 LLM Planning Baseline Comparison

As shown in Table 1, LLM-REGRESS significantly outperforms LLM baselines across all three benchmarks in success rate, input tokens, and duration. LLMs reliably generate affordances for common household items. However, we observe that sometimes LLMs produce affordances, while aligned with commonsense, that are not supported by the benchmark (e.g., `canCool(plate, bread)`), indicating the need for improved affordances constraints in existing benchmarks. All methods show performance deterioration on the ALFWorld-Afford benchmark as task complexity increases. For Tabletop-Afford, LLM-REGRESS still achieves the best performance, though the gap narrows, likely due to its full observability. LLM-REGRESS is also more efficient in token usage and planning duration since it generates predicate-level affordances only when necessary. Finally, we note that LLMs cannot directly leverage the action model used in LLM-REGRESS and instead rely on past examples as plan generation templates.

5.2 Grounded Planning Baseline Comparison

Grounded planners require complete problem specification to generate a plan. Consequently, an LLM must exhaustively enumerate affordances facts to construct both the domain and problem specifications for the planner, which amounts to approximately $4,000$ tokens per task in ALFWorld. This significantly increases the likelihood of hallucination, including syntax errors and infeasible affordances, ultimately leading to plan failure. In contrast, LLM-REGRESS generates affordances information only when necessary for a lifted plan, greatly reducing the chances of hallucinations. As a result, we observe that grounded planners relying on LLM-generated affordances perform significantly worse than LLM-REGRESS.

5.3 Multi-Trial Baseline Comparison

As shown in Figure 3, we observe a monotonic increase in performance when LLM-REGRESS utilizes the knowledge from prior trials. Specifically, LLM-REGRESS achieved significantly better performance as more affordance knowledge was transferred, reaching respective success rates of 99%, 91%, and 91% in ALFWorld, ALFWorld-Afford, and Tabletop-Afford after 3 trials. As for Reflexion and Direct LLM, LLMs need to extract useful information via text and determine what information is useful (e.g., affordances information versus room layout details). In contrast, our structured approach explicitly tracks and stores relevant facts in the KB that are only used if necessary for a subgoal.

Table 2: Success rate performance comparison of the impact of affordances knowledge and different LLM choices on our method across benchmarks.

Method	**ALFWorld**	**ALFWorld -Afford**	**Tabletop-Afford**
No-Affordances	0.12	0.05	0.21
LLM-Affordances			
w/ GPT-3.5-Turbo	0.91	0.73	0.70
w/ GPT-4o	0.96	0.81	0.83
Perfect-Affordances	1.00	0.98	1.00

5.4 Ablation

In this section, we present an ablation study evaluating the impact of affordances knowledge and LLM choice on agent performance. As Table 2 shows the **Perfect-Affordances** method provides ground-truth affordances information, demonstrating the performance upper bound. Results show that affordances generated by `GPT-4o` outperform those from `GPT-3.5-Turbo`, indicating that a better language model enhances affordances reasoning. In the **No-Affordances** setting, all actions are applicable to all objects, rendering most plans infeasible. This underscores the critical importance of affordances reasoning, as without it, the agent cannot generate reasonable plans or select appropriate actions. Other failure modes are included in a detailed analysis provided in Appendix N.

6 Conclusion

In this work, we introduced LLM-REGRESS, an open-world planning approach that integrates *lifted regression* with LLM-generated affordances. LLM-REGRESS generates sound and complete plans in a compact lifted representation, reducing the need for exhaustive state and action enumeration, while leveraging LLMs to infer goal-relevant objects and affordances without requiring full domain specification. Our experiments on three benchmarks demonstrate that LLM-REGRESS significantly outperforms state-of-the-art LLM planners and grounded forward planner with LLM-generated affordances. The results highlight LLM-REGRESS as a scalable and generalizable solution for open-world planning, paving the way for more robust and interpretable embodied AI systems.

Acknowledgments

This work was supported by LG Electronics, Toronto AI Lab Grant Ref No. 2024-0565.

Limitations

While our approach demonstrates strong performance on existing embodied AI benchmarks, our focus on high-level embodied agent planning introduces several limitations. First, our experiments are conducted in deterministic environments, which may require further adaptation for real-world robotics. Second, the natural language instructions and environment observations are generated within a simulator, making them less noisy than human-annotated data. Finally, although LLM-Regress is highly token-efficient compared to existing LLM-based approaches, it still assumes access to large-scale language models such as `GPT-4o`. While LLM-Regress can be readily adapted to extract affordance information from structured knowledge bases and knowledge graphs in computationally constrained settings, this lies beyond the scope of the current work.

Ethical Statement

LLM-REGRESS leverages lifted regression and LLM-based commonsense reasoning for open-world planning. While our approach is theoretically sound and complete, offering an interpretable symbolic model and plans, its evaluation has been confined to simulation environments. Transitioning these methods to real-world applications will require additional ethical considerations regarding testing and deployment. Moreover, because our approach relies on LLMs for generating affordances, an area prone to hallucinations and unpredictable behavior, careful monitoring, rigorous testing, and continual ethical review are imperative to ensure safe operation and alignment with societal values.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

Daman Arora and Subbarao Kambhampati. 2023. Learning and leveraging verifiers to improve planning capabilities of pre-trained language models. *CoRR*, abs/2305.17077.

Diego Calvanese, Giuseppe De Giacomo, Marco Montali, and Fabio Patrizi. 2018. First-order μ-calculus over generic transition systems and applications to the situation calculus. *Information and Computation*, 259:328–347.

Guanqi Chen, Lei Yang, Ruixing Jia, Zhe Hu, Yizhou Chen, Wei Zhang, Wenping Wang, and Jia Pan. 2024. Language-augmented symbolic planner for open-world task planning. *arXiv preprint arXiv:2407.09792*.

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

Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated Planning: theory and practice*. Elsevier.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. In *NeurIPS*.

Marc Hanheide, Moritz Göbelbecker, Graham S Horn, Andrzej Pronobis, Kristoffer Sjöö, Alper Aydemir, Patric Jensfelt, Charles Gretton, Richard Dearden, Miroslav Janicek, et al. 2017. Robot task planning and explanation in open and uncertain worlds. *Artificial Intelligence*, 247:119–150.

Rishi Hazra, Pedro Zuidberg Dos Martires, and Luc De Raedt. 2024. Saycanpay: Heuristic planning with large language models using learnable domain knowledge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20123–20133.

Malte Helmert. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246.

Jörg Hoffmann and Bernhard Nebel. 2001. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232*.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR.

Yuqian Jiang, Nick Walker, Justin Hart, and Peter Stone. 2019. Open-world reasoning for service robots. In *Proceedings of the international conference on automated planning and scheduling*, volume 29, pages 725–733.

Yongmei Liu and Gerhard Lakemeyer. 2010. On first-order definability and computability of progression for local-effect actions and beyond. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Vittorio Perera, Robin Soetens, Thomas Kollar, Mehdi Samadi, Yichao Sun, Daniele Nardi, René Van de Molengraft, and Manuela Veloso. 2015. Learning task knowledge from dialog and web access. *Robotics*, 4(2):223–252.

Raymond Reiter. 2001. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT press.

Scott Sanner and Craig Boutilier. 2009. Practical solution techniques for first-order mdps. *Artificial Intelligence*, 173(5-6):748–788.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. 2022. Cliport: What and where pathways for robotic manipulation. In *Conference on robot learning*, pages 894–906. PMLR.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Tom Silver and Rohan Chitnis. 2020. Pddlgym: Gym environments from pddl problems. *arXiv preprint arXiv:2002.06432*.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. 2023. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2998–3009.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36:75993–76005.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, et al. 2023. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *arXiv preprint arXiv:2311.05997*.

Zihao Wang, Anji Liu, Haowei Lin, Jiaqi Li, Xiaojian Ma, and Yitao Liang. 2024. Rat: Retrieval augmented thoughts elicit context-aware reasoning in long-horizon generation. *arXiv preprint arXiv:2403.05313*.

Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. 2023a. Translating natural language to planning goals with large-language models. *CoRR*, abs/2302.05128.

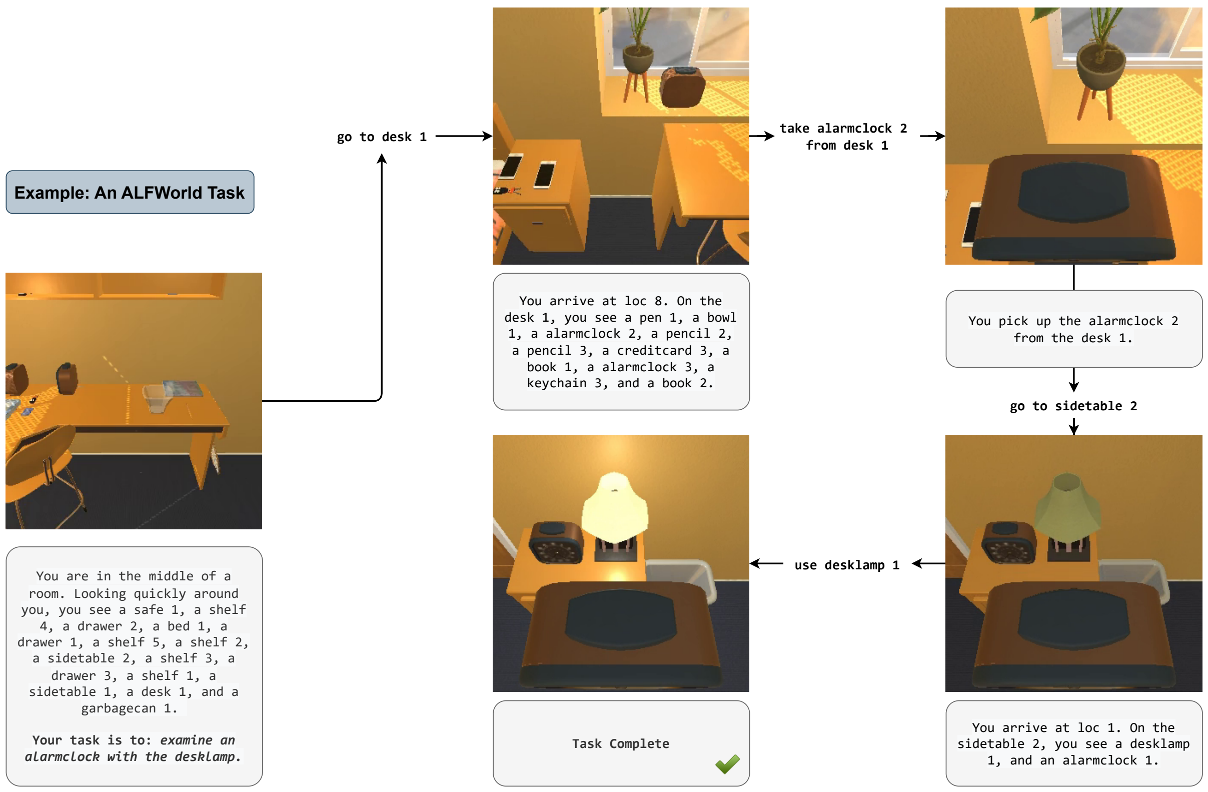
Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. 2023b. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, page 100211.

Appendix

A Illustrative Example: An ALFWorld Task



You are in the middle of a room. Looking quickly around you, you see a safe 1, a shelf 4, a drawer 2, a bed 1, a drawer 1, a shelf 5, a shelf 2, a sidetable 2, a shelf 3, a drawer 3, a shelf 1, a sidetable 1, a desk 1, and a garbagecan 1.

Your task is to: *examine an alarmclock with the desklamp.*

go to desk 1

You arrive at loc 8. On the desk 1, you see a pen 1, a bowl 1, a alarmclock 2, a pencil 2, a pencil 3, a creditcard 3, a book 1, a alarmclock 3, a keychain 3, and a book 2.

take alarmclock 2 from desk 1

You pick up the alarmclock 2 from the desk 1.

go to sidetable 2

You arrive at loc 1. On the sidetable 2, you see a desklamp 1, and an alarmclock 1.

use desklamp 1

Task Complete

Figure 4: **Illustrative Example for a Given ALFWorld Task**: In this example, the agent is tasked with "Examine an alarm clock under the desklamp". To complete the task, the agent must first locate an alarm clock and pick it up, then find a desk lamp and turn it on. The agent initially does not observe the alarm clock and desk lamp and needs to actively explore the environment to find objects that can fulfill its goal.

B LLM-REGRESS Example for An ALFWorld Task

Here is a successful planning example for our LLM-REGRESS method for an ALFWorld task.

```
    -= Welcome to TextWorld, ALFRED! =-

You are in the middle of a room. Looking quickly around you, you see
a cabinet 6, a cabinet 5, a cabinet 4, a cabinet 3, a cabinet 2,
a cabinet 1, a coffeemachine 1, a countertop 3, a countertop 2,
a countertop 1, a drawer 3, a drawer 2, a drawer 1, a fridge 1,
a garbagecan 1, a microwave 1, a shelf 3, a shelf 2, a shelf 1,
a sinkbasin 1, a stoveburner 4, a stoveburner 3, a stoveburner 2,
a stoveburner 1, and a toaster 1.

Your task is to: heat some egg and put it in garbagecan.
go to stoveburner 2
you arrive at loc 19. on the stoveburner 2, you see a pan 1.
go to microwave 1
you arrive at loc 17. the microwave 1 is closed.
go to garbagecan 1
you arrive at loc 31. on the garbagecan 1, you see a soapbottle 2, and a tomato 3.
go to stoveburner 3
you arrive at loc 15. on the stoveburner 3, you see a pot 1.
go to shelf 2
you arrive at loc 3. on the shelf 2, you see a creditcard 3, and a saltshaker 1.
go to fridge 1
you arrive at loc 30. the fridge 1 is closed.
go to cabinet 3
you arrive at loc 20. on the cabinet 3, you see nothing.
go to stoveburner 1
you arrive at loc 15. on the stoveburner 1, you see a pot 1.
```

```
go to cabinet 4
you arrive at loc 21. on the cabinet 4, you see nothing.
go to cabinet 2
you arrive at loc 11. the cabinet 2 is closed.
open cabinet 2
you open the cabinet 2. the cabinet 2 is open. in it, you see nothing.
open microwave 1
you open the microwave 1. the microwave 1 is open. in it, you see nothing.
go to drawer 1
you arrive at loc 28. the drawer 1 is closed.
open drawer 1
you open the drawer 1. the drawer 1 is open. in it, you see a peppershaker 2.
go to countertop 3
you arrive at loc 5. on the countertop 3, you see a butterknife 3,
a butterknife 2, a butterknife 1, a cellphone 1, a cup 2, a egg 1,
a fork 1, a houseplant 1, a lettuce 3, a plate 2, a plate 1,
a potato 1, a tomato 2, and a vase 3.

LLM Prompt: I am an embodied agent, I have observed a list of objects:
cabinet_6 , cabinet_5 , cabinet_4 , cabinet_3 , cabinet_2 , cabinet_1
, coffeemachine_1 , countertop_3 , countertop_2 , countertop_1 ,
drawer_3 , drawer_2 , drawer_1 , fridge_1 , garbagecan_1 , microwave_1
, shelf_3 , shelf_2 , shelf_1 , sinkbasin_1 , stoveburner_4 ,
stoveburner_3 , stoveburner_2 , stoveburner_1 ,
toaster_1 I want to the best object or receptalce ['R']

 from my observations, so that this fact is True:
 inreceptacle(countertop_3, egg_1) & isgarbagecan(garbagecan_1)
 & isegg(egg_1)
 & canheat(R, egg_1)
. Please give me the best object or receptacle that would satisfy my objective.
. Please give the answer in format like best_answer:(obj_1)

LLM answer: best_answer:(microwave_1)
llm generated affordances: canheat(R, egg_1) ['microwave_1']
take egg 1 from countertop 3
you pick up the egg 1 from the countertop 3.
heat egg 1 with microwave 1
you heat the egg 1 using the microwave 1.
put egg 1 in/on garbagecan 1
you put the egg 1 in/on the garbagecan 1.
Success: True
```

C Goal Parser

We parse the natural language goal to a symbolic form using action models and few-shot examples. The prompt that we use is shown below:

```
Action Model: {
(:action pick-up
    :parameters (?x - obj ?r - robot)
    :precondition (and
                  (canPickUp ?x)
                  (clear ?x)
                  (onTable ?x)
                  (handEmpty ?r)
                 )
    :effect (and
            (not (onTable ?x))
            (not (clear ?x))
            (not (handEmpty ?r))
            (handFull ?r)
            (holding ?x)
           )
)

(:action put-down
    :parameters (?x - obj ?r - robot)
    :precondition (and
                  (holding ?x)
                  (handFull ?r)
                 )
    :effect (and
            (not (holding ?x))
            (clear ?x)
            (handEmpty ?r)
            (not (handFull ?r))
            (onTable ?x)
           )
)
.....................
{
  "Example 1": {
    "Language Goal": "Put a hot potato on the countertop.",
    "Symbolic Goal": "isHot(?x), isPotato(?x), isCountertop(?y), inReceptacle(?x, ?y)"
  },
  "Example 2": {
    "Language Goal": "Put a hot cup in the fridge.",
    "Symbolic Goal": "isHot(?x), isCup(?x), isFridge(?y), inReceptacle(?x, ?y)"
  }
}

Generate symbolic goal from language goal: Put a hot apple on a plate

Your results should be in the form of {<symbolic goal>}
```

D LLM Object Parser Prompt

```
LLM Prompt: I am an embodied agent, My observations is "{}"
I want to extract list of objects to observe.
Please give me a set of objects in this format: {obj_1, obj_2, obj_3}
```

E LLM Affordances Query Prompt

```
LLM Prompt: I am an embodied agent,
I have observed a list of objects: {}
I want the object or receptacle from my observations, so that this fact is True:
KB: {}
Please give me the best object or receptacle that would satisfy my objective.
Please give the answer in format like best_answer: (obj_1)
```

F Grounded Forward Search W/ LLM Affordances Prompt

```
LLM Prompt:
Given
A PDDL domain: {*content of domain file}
A PDDL problem: {*content of problem file with removed affordances}
An agent is tasked to {PDDL Goal, Natural Language Goal}

Please generate affordances required to complete the task, you should use PDDL
syntax specified in PDDL domain and PDDL problem
```

G The STRIPS-Style Action Model for ALFWorld and ALFWorld-Afford

```
(:action PutObjectInReceptacle
    :parameters (?o - obj ?r - obj)
    :precondition (and
        (holds ?o)
        (holdsAny)
        (not (isContained ?o))
    )
    :effect (and
        (inReceptacle ?r ?o)
        (isContained ?o)
        (not (holds ?o))
        (not (holdsAny))
    )
)
```

```
(:action HeatObject
    :parameters (?r - obj ?o - obj)
    :precondition (and
        (canHeat ?r ?o)
        (holds ?o)
        (holdsAny)
        (not (isContained ?o))
    )
    :effect (and (isHot ?o))
)

(:action CleanObject
    :parameters (?r - obj ?o - obj)
    :precondition (and
        (canClean ?r ?o)
        (holds ?o)
        (holdsAny)
        (not (isContained ?o))
        (not (isHot ?o))
    )
    :effect (and (isClean ?o))
)

(:action ToggleObject
    :parameters (?o - obj)
    :precondition (and
        (canToggle ?o)
        (holds ?o)
        (holdsAny)
        (not (isContained ?o))
    )
    :effect (and (isOn ?o))
)
```

H The STRIPS-Style Action Model for Tabletop-Afford

```
(:action pick-up
  :parameters (?x - obj ?r - robot)
  :precondition (and
                (canpickup ?x)
                (clear ?x)
                (ontable ?x)
                (handempty ?r)
               )
  :effect (and
          (not (ontable ?x))
          (not (clear ?x))
          (not (handempty ?r))
          (handfull ?r)
          (holding ?x)
         )
)

(:action put-down
  :parameters (?x - obj ?r - robot)
  :precondition (and
                (holding ?x)
                (handfull ?r)
               )
  :effect (and
          (not (holding ?x))
          (clear ?x)
          (handempty ?r)
          (not (handfull ?r))
          (ontable ?x)
         )
)
```

```
(:action stack
  :parameters (?x - obj ?y - obj ?r - robot)
  :precondition (and
                (canstack ?x ?y)
                (holding ?x)
                (clear ?y)
                (handfull ?r)
                )
  :effect (and
          (not (holding ?x))
          (not (clear ?y))
          (clear ?x)
          (handempty ?r)
          (not (handfull ?r))
          (on ?x ?y)
          )
)

(:action unstack
  :parameters (?x - obj ?y - obj ?r - robot)
  :precondition (and
                (canpickup ?x)
                (on ?x ?y)
                (clear ?x)
                (handempty ?r)
                )
  :effect (and
          (holding ?x)
          (clear ?y)
          (not (clear ?x))
          (not (handempty ?r))
          (handfull ?r)
          (not (on ?x ?y))
          )
)
```

I The ALFWorld-Afford Benchmark

We created 150 additional tasks on top of the text version of ALFWorld, including 5 new task types, each requiring at least two affordances. Additional affordances for new objects were also added to increase diversity.

I.1 Number of Tasks and Affordances Actions

Task	Number of Tasks	Affordances Actions
pick-clean-heat-put	20	Heat, Clean
pick-clean-cook-put	20	Cool, Clean
pick-heat-cool-put	20	Heat, Cool
pick-clean-heat-put-toggle	20	Heat, Clean, Toggle
pick-clean-cool-put-toggle	20	Cool, Clean, Toggle

Table 3: Number of tasks and affordances actions in ALFWorld-Afford.

I.2 Extra Affordances

Additional affordances for new objects include:

- **Heat:** Toaster {Bread}, Coffee Machine {Mug}, Stove Burner {Pan, Pot}
- **Cool:** Countertop {Cup, Plate, Pan, Bowl}
- **Clean:** Cloth {Apple, Egg, Cup, Pan, Tomato}, Dish Sponge {Cup, Plate, Pan, Bowl}
- **Toggle:** Microwave, Faucet, Laptop, Light Switch, Television, Cellphone, Toaster, Stove Burner

J The Tabletop-Afford Benchmark

An example task translated to language instruction is shown below:

```
You are in the middle of a countertop. You see a plate, a cup, a bowl, a can, a fork, a knife, a
spoon, a napkin, a book, a remote, a phone, a pen, a key, a bottle, an apple, a banana, a chair,
and a laptop.
Your task is to: stack the can on the bowl, the bowl on the plate, and the plate on the book.
```

The list of objects is: {plate, cup, bowl, can, fork, knife, spoon, napkin, book, remote, phone, pen, key, bottle, apple, banana, mug, glass, vase, clock, scissors, towel, pillow, newspaper, cereal_box, cheese, milk, bread, chair, laptop}, and we randomly choose a subset of the items and their location for the tasks.

K First Order Logic Operations

Our approach relies on key logical operators to facilitate lifted regression and ensure sound plan generation.

- **Unification**: The unification operator UNIFY(p, q) checks whether two sets of predicates with different variables are equivalent. If they can be unified, it returns a substitution θ:

$$\text{UNIFY}(p, q) = \theta \quad \text{where} \quad \theta(p) = \theta(q).$$

  The substitution function θ is a most general unifier (MGU), ensuring that two predicates are equivalent.
- **Standardization**: The standardization operator STANDARDIZE(p) replaces all variables in p with fresh variables $v' \notin \mathcal{V}$ such that:

$$\forall v \in \text{vars}(p), v \notin \text{vars}(\text{STANDARDIZE}(p)).$$

  Standardization prevents variable name conflicts when reusing action schemas in a plan.
- **Substitution**: A substitution θ is a mapping from variables to terms, denoted as:

$$\theta = \{x_1/t_1, x_2/t_2, ..., x_n/t_n\}.$$

  Applying θ to a predicate p, written $\theta(p)$, replaces each variable x_i in p with its corresponding term t_i. Substitutions are used to instantiate lifted actions and query the LLM predicates in regression.

L LLM Regression Reasoner

The LLM-REGRESS reasoner utilizes a set of *LiftedPlans* generated via Open-World Lifted Regression, represented as a set of tuples (g, π), where g denotes a subgoal and π its corresponding plan. Before executing any action, for each $(g, \pi) \in$ LiftedPlans, the algorithm identifies affordances predicates as:

$$g_a = \{p \in g \mid \text{type}(p) \in \text{AffordPredTypes}\}$$

and defines non-affordances predicates as:

$$g_{na} = g \setminus g_a.$$

If the agent's KB satisfies g_{na}, we then obtain the set of satisfying facts F_{na} by querying the agent's KB. The affordances predicates F_a are generated by querying LLM:

$$F_a \leftarrow \text{LLMAffQuery}(F_{na}, \text{FailedAff}, O)$$

Once the entire subgoal g is satisfied, each action $a \in \pi$ is executed sequentially. If an action fails, the corresponding g_a is added to FailedAff, and the agent removes it from the knowledge base. Otherwise, the agent updates its KB with respect to a via:

$$\text{PROGRESS}(\text{KB}, a) = \text{KB} \cup \text{add}(a) \setminus \text{del}(a).$$

If no feasible plan can be found, the agent explores the environment randomly. The object set O is augmented with new objects parsed by the *LLM Object Parser* during exploration. This process repeats until the overall goal G is fully satisfied in the knowledge base.

M Success Rate Breakdown by Task Type

In this section, we present the performance breakdown for ALFWorld and ALFWorld Afford. The following table shows the performance for each task type in terms of success rate.

Table 4: Success Rate Breakdown by Task Category (ALFWorld)

Task Category	LLM-Regress	Direct LLM (GPT-4o)	ReAct	GroundedPlanner
Pick & Place	1.00	0.20	0.40	0.36
Examine in Light	0.98	0.15	0.85	0.12
Clean & Place	0.96	0.30	0.90	0.27
Heat & Place	0.98	0.25	0.75	0.53
Cool & Place	0.94	0.10	0.87	0.22
Pick Two & Place	0.84	0.26	0.43	0.60
Overall	**0.95**	**0.21**	**0.70**	**0.35**

Table 5: Success Rate Breakdown by Task Category (ALFWorld-Afford)

Task Category	LLM-Regress	Direct LLM (GPT-4o)	ReAct	GroundedPlanner
pick-clean-heat-put	1.00	0.10	0.55	0.35
pick-clean-cool-put	0.90	0.15	0.65	0.35
pick-heat-cool-put	0.85	0.10	0.60	0.20
pick-clean-heat-put-toggle	0.70	0.10	0.40	0.25
pick-clean-cool-put-toggle	0.75	0.20	0.65	0.30
Overall	**0.84**	**0.13**	**0.57**	**0.29**

N Failure Analysis

We provide a breakdown of failure modes for each benchmark in this section, where the reported numbers indicate the proportion of failures attributed to each category. This analysis is conducted only on failure cases, which represent a small subset of the total tasks evaluated. The failure categories include: Incorrect Affordance, referring to failures caused by inaccurately generated affordances; Insufficient Exploration, where the agent fails to locate all necessary objects related to the goal; Goal Parsing Error, due to discrepancies between parsed and ground-truth goals; Observation Parsing Error, where objects in the observation are misinterpreted; and Other, which includes syntax errors from LLMs, incorrect action formats, simulator issues, and other miscellaneous problems.

Table 6: Failed Tasks Breakdown by Failure Modes

Failure Mode	ALFWorld	ALFWorld-Afford	TableTop-Afford
Incorrect Affordances	0.22	0.35	0.51
Insufficient Exploration	0.32	0.33	Not Applicable
Goal Parsing Error	0.11	0.08	0.12
Observation Parsing Error	0.21	0.17	0.11
Others	0.14	0.07	0.26