NATURAL LANGUAGE PDDL (NL-PDDL) FOR OPEN-WORLD GOAL-ORIENTED COMMONSENSE REGRESSION PLANNING IN EMBODIED AI

Anonymous authors
Paper under double-blind review

ABSTRACT

Planning in open-world environments, where agents must act with partially observed states and incomplete knowledge, is a central challenge in embodied AI. Open-world planning involves not only sequencing actions but also determining what information the agent needs to sense to enable those actions. Existing approaches using Large Language Models (LLM) and Vision-Language Models (VLM) cannot reliably plan over long horizons and complex goals, where they often hallucinate and fail to reason causally over agent-environment interactions. Alternatively, classical PDDL planners offer correct and principled reasoning, but fail in open-world settings: they presuppose complete models and depend on exhaustive grounding over all objects, states, and actions; they cannot address misalignment between goal specifications (e.g., "heat the bread") and action specifications (e.g., "toast the bread"); and they do not generalize across modalities (e.g., text, vision). To address these core challenges: (i) we extend symbolic PDDL into a flexible natural language representation that we term NL-PDDL, improving accessibility for non-expert users as well as generalization over modalities; (ii) we generalize regression-style planning to NL-PDDL with commonsense entailment reasoning to determine what needs to be observed for goal achievement in partially-observed environments with potential goal–action specification misalignment; and (iii) we leverage the lifted specification of NL-PDDL to facilitate open-world planning that avoids exhaustive grounding and yields a time and space complexity independent of the number of ground objects, states, and actions. Our experiments in three diverse domains — classical Blocksworld and the embodied ALFWorld environment with both textual and visual states — show that NL-PDDL substantially outperforms existing baselines, is more robust to longer horizons and more complex goals, and generalizes across modalities.

1 INTRODUCTION

Open-world planning[1], where the agent operates under partial observability and incomplete knowledge, is essential for embodied agents to perform real-world tasks. Such embodied environments are inherently open-world and involve reasoning over a multitude of objects, posing challenges for planning. For example, we consider Figure 1, where the agent is asked to *"toast the bread and leave it on a plate"*. To efficiently identify the sequence of actions that achieve this goal given only partial observations of the state and an incomplete environment model, the agent must find the relevant objects (e.g., *bread*, *toaster*, and *plate*) while judiciously ignoring irrelevant ones. It must also use its commonsense knowledge to reason that a *toaster can heat bread*, but *a pot of water cannot*.

The breakthrough of foundation models such as Large Language Models (LLM) and Vision-Language Models (VLM) has led to their increased usage in planning (Yao et al., 2023b; Li et al., 2024; Kong et al., 2024); yet, they come with critical limitations (Shojaee et al., 2025; Kambhampati et al., 2024). These models struggle to generate reliable long-horizon plans with sound causal reasoning about the interactions between the agent and its environment as they lack a mechanism to

[1]In this work, we characterize *open-world* by partial observability and incomplete domain knowledge, distinguishing it from *open-world games*, which refers to open-ended tasks in expansive environments.

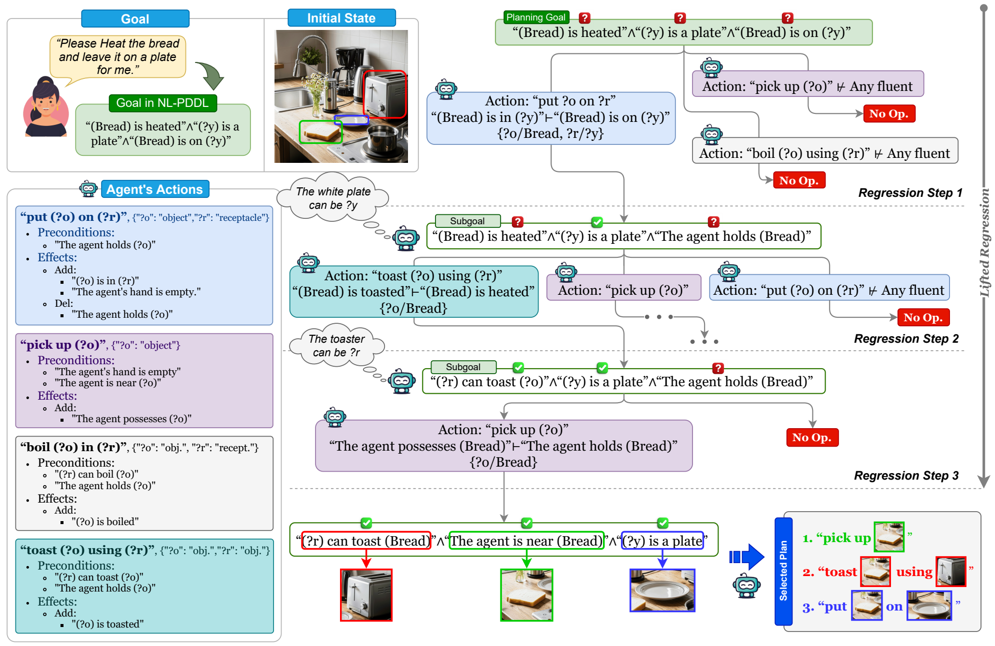

Figure 1: **Overview of the NL-PDDL framework:** (**Agent's Actions**) The specification of actions available to the agent are provided in a natural language (NL) variant of PDDL that we introduce. (**Goal**) The user provides NL instructions that are automatically translated into an NL-PDDL goal specification. (**Initial State**) The initial state is given as an image. (**Planning Goal**) Open-world first-order planning proceeds top-down: starting from the goal, regressed subgoals are generated that must hold before each action can be applied (e.g., the goal *(bread) is heated* regresses to the action *toast (bread) using (toaster)*, which generates subgoals *the agent holds the (bread)* and *a toaster is available*). At each step, LLM-based entailment connects subgoals with the NL effects of candidate actions and reasons about object affordances (e.g., *a toaster can toast bread, but a pot of water cannot*). (**Selected Plan**) A VLM grounds the NL object names in a regressed subgoal to their corresponding entities in the initial state image that are then used to instantiate an actionable plan.

track state changes and project how actions alter the world (Valmeekam et al., 2024). Moreover, they fail on complex goals involving multiple logical constraints (Goebel & Zips, 2025), and their black-box nature makes it hard to interpret or verify their generated plans (Aghzal et al., 2025). In contrast, classical planners, such as those based on the Planning Domain Definition Language (PDDL) (Aeronautiques et al., 1998), have traditionally served the role of planning in AI. While verifiably correct, these methods often presume all objects and relations are known in a perfect model that can be exhaustively grounded. These methods also cannot bridge misalignments between goals and action specifications (e.g., reasoning that the action *"toast the bread"* is able to achieve the goal *"heat the bread"*a rev). Such limitations are too restrictive for the the open-world setting we address here.

To overcome the limitations of existing methods, we introduce NL-PDDL, a planning framework that combines the expressive flexibility of natural language (NL) with the formal guarantees of symbolic planning. An overview of the NL-PDDL workflow is provided in Figure 1 and its caption. Concretely, we make the following key contributions to open-world planning in embodied AI:

- We extend classical PDDL into NL-PDDL, a flexible representation that lets users specify goals and actions abstractly in NL. This flexibility reduces the specification burden (vs. rigid PDDL schemas) and tolerates semantically incomplete and syntactically imperfect domain and state descriptions while maintaining sound plan generation.
- We propose an open-world, regression-style NL-PDDL planner that uses LLM-based commonsense entailment over regressed subgoals to both infer the observations needed to achieve a target (e.g., *heat bread* → *find toaster*) and resolve misalignments between (sub)goals and action effects.

- We avoid exhaustive grounding via *lifted regression* (Reiter, 1991). Instead of reasoning about specific objects (e.g., *pan3*, *toaster1*), lifted regression leverages the variables inherent in NL-PDDL (e.g., *"there exists some x that can toast bread"*). Lifted plans are instantiated only when suitable objects are found, making planning complexity independent of the number of objects and abstractly capturing a lifted *conditional plan* of all initial state conditions that can achieve a goal.
- Across three diverse planning domains — Blocksworld (Valmeekam et al., 2022), ALFworld, and ALFworld-Vision (Shridhar et al., 2021) — we show that lifted regression planning in NL-PDDL achieves higher plan-success rates than strong baselines, remains robust as plan horizons increase, and generalizes across both text and vision modalities.

2 Methodology

In this section, we present Natural Language PDDL (**NL-PDDL**), a framework for open-world, goal-oriented planning by leveraging regression-style reasoning enhanced with LLM-based commonsense reasoning. While classical PDDL provides a sound framework for long-horizon planning, it cannot facilitate commonsense reasoning unless such knowledge is explicitly encoded by a domain designer. For example, a classical PDDL solver cannot infer that `isToasted(bread)` implies `isHeated(bread)`, even though it is intuitively clear to humans that *"if bread is toasted, it is likely heated"*. NL-PDDL builds on the formal framework of classical PDDL but replaces symbolic predicates, variables, and objects with typed NL counterparts, enabling seamless integration of LLMs to perform commonsense entailment inferences during planning. For a formal review of PDDL and first-order regression planning extended here, we refer the reader to Appendix B.

Natural Language Representation of PDDL. In classical PDDL, predicates, objects, and variables are represented with rigid symbols. NL-PDDL replaces these symbols with typed NL counterparts. For instance, the symbolic predicate `isToasted(bread)` becomes *"the (bread) is toasted"*|*"bread"*:*"food"*, while *"the (?o:food) is toasted"* denotes its lifted form. Here, *?o:food* indicates that *(?o)* is of type *food*.

NL-PDDL Problem Definition. NL-PDDL problems are defined like classical PDDL problems, but aimed at open-world planning. We define an NL-PDDL problem as a tuple $\mathcal{P} = \langle G, \mathcal{A}, \mathcal{F}, h \rangle$, where G is a conjunctive formula expressing the agent's goal, and $\mathcal{F}$ is the set of lifted NL predicates in the domain. $\mathcal{A}$ is the set of actions $a(\vec{y})$, i.e., parameterized first-order operators. Each $a(\vec{y}) \in \mathcal{A}$ is defined by: preconditions $a(\vec{y})$.pre $\subseteq \mathcal{F}$, which must hold for the action to execute; add effects $a(\vec{y})$.add $\subseteq \mathcal{F}$, which become true after the action is executed; and delete effects $a(\vec{y})$.del $\subseteq \mathcal{F}$, which no longer hold after execution. The planning horizon $h \in \mathbb{N}$ is the maximum length of feasible plans. The goal in NL-PDDL is to construct a *conditional plan* Π that maps subgoal formulas to action sequences that can achieve them:

$$\Pi = \left[(\psi_1(\vec{x_1}), \langle a^{1,1}(\vec{x^{1,1}}), ..., a^{1,n_1}(\vec{x^{1,n_1}})\rangle), ..., (\psi_k(\vec{x_k}), \langle a^{k,1}(\vec{x^{k,1}}), ..., a^{k,n_k}(\vec{x^{k,n_k}})\rangle)\right], \quad (1)$$

where each $\psi_i(\vec{x_i})$ is a first-order formula representing a subgoal, and $\langle a^{i,1}(\vec{x^{i,1}}), \ldots, a^{i,n_i}(\vec{x^{i,n_i}})\rangle$ is the corresponding sequence of actions (i.e., a plan) with length $n_i \leqslant h$, and each $\vec{x^{j,k}} \subseteq \vec{x^j}$. As an example, the plan derived by NL-PDDL for the problem in Figure 1 is presented in Figure 2.

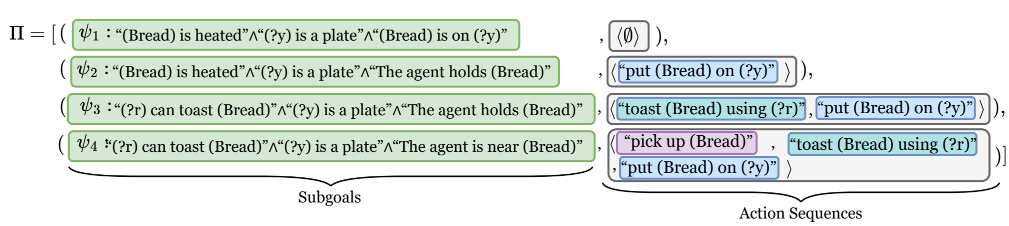

Figure 2: The conditional plan derived by NL-PDDL for the problem in Figure 1.

2.1 First Order Regression in NL-PDDL

Effect Axioms. Following the methodology for first-order regression defined in Appendix B, we first need to transform our NL-PDDL description into the *effect axioms* that are used to perform regression planning. Specifically, following the notation of Appendix B, we build axioms of the form of Equations 8 and 9, and provide a concrete example of $\gamma^{+}_{\text{“(?o) is toasted”},\text{“toast(?o) using (?r)”}}(?o, ?r)$ for action *"toast(?o) using (?r)"*, and predicate *"(?o) is toasted"* from the NL-PDDL description in Figure 3. In English, this says an object *"(?o)"* is toasted in the next state if the agent holds

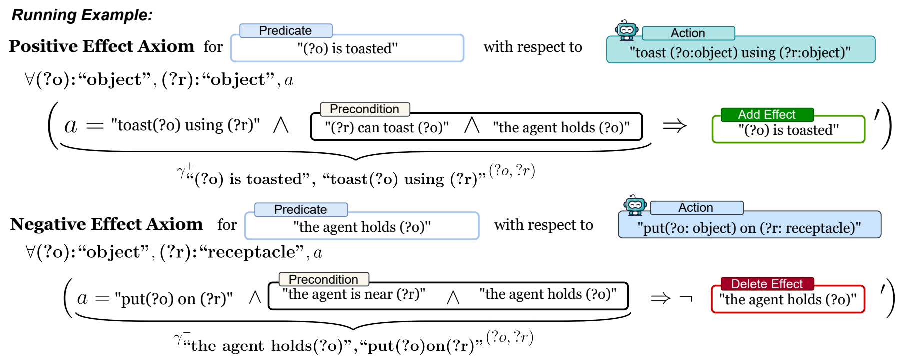

Figure 3: Example of positive and negative effect axiom based on NL-PDDL problem in Figure 1.

it, and the agent toasts *"(?o)"* using object *"(?r)"* that can toast *"(?o)"*. To illustrate negative effect axioms, we construct $\gamma^{-}_{\text{“the agent holds(?o)”},\text{“put(?o) on (?r)”}}(?o, ?r)$ for action *"put(?o) on (?r)"* and predicate *"the agent holds(?o)"* in Figure 3. In English, this says if the agent holds an object *"(?o)"*, is near receptacle *"(?r)"*, and puts *"(?o)"* on *"(?r)"*, then the agent no longer holds *"(?o)"* in the next state.

These positive (negative) effect axioms are only constructed for predicates and actions if the predicate is in the add (delete) effect of the action in the NL-PDDL description. Cases that a predicate appears in the effects of multiple actions are handled via disjunction (see Appendix C for details).

Successor State Axioms. As described in Appendix B, successor state axioms (SSAs) are the workhorse of lifted regression planning in that they allow the replacement of a predicate in a post-action state with the necessary subgoals required to achieve it. Let $F'_{a_i}(\vec{y_F})$ denote the value of a predicate $F(\vec{y_F})$ in the next state after executing action $a_i(\vec{y})$. The SSA for $F(\vec{y_F})$ is defined as:

$$F'_{a_i}(\vec{x}) \equiv \gamma^{+}_{F,a_i}(\vec{x}) \;\vee\; \big(F(\vec{x}) \wedge \neg\gamma^{-}_{F,a_i}(\vec{x})\big). \tag{2}$$

In English, an SSA states that a predicate F is true in the successor state *iff* it was made true by a positive effect, or it was already true and not made false by a negative effect.

We construct the SSAs for each action-predicate pair. While we leave complete formal details to the Appendix B due to space limitations, we provide the SSA for the predicate *"is toasted(?o)"* and action *"toast (?o) using (?r)"*, and the predicate *"the agent possesses(?o)"* and action *"put (?o) on (?r)"* in Figure 4 as examples to illustrate the process of SSA construction:

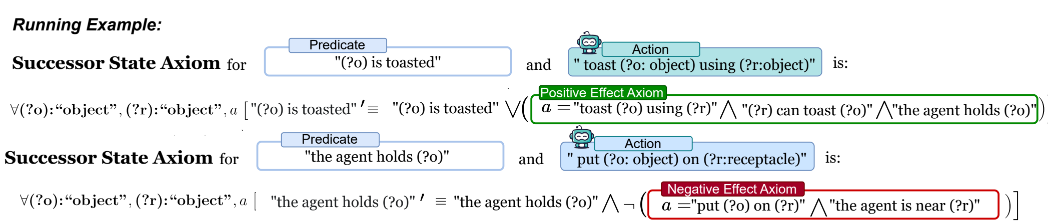

Figure 4: Examples of constructing SSAs based on NL-PDDL problem in Figure 1.

After constructing the SSAs, NL-PDDL recursively regresses each subgoal through applicable actions to construct the conditional plan Π. We explain the regression procedure in the next section.

2.1.1 Regression of Potentially Misaligned Predicates in NL-PDDL

In regression planning with classical PDDL, goal predicates are only matched to action effects that share the same predicate names. In NL, such exact matches rarely occur due to variability in phrasing, instead an action effect may entail a goal literal, e.g., "the bread is toasted" entails ($\vdash$) "the bread is heated". In this section, we extend lifted regression planning using LLM entailment ($\vdash_{\text{LLM}}$).

Regression of Positive NL Predicates. Let $P'(\vec{x})|\vec{x}:\vec{T_x}$ be a predicate in the next state. We aim to regress it through an action a_i, to derive pre-action conditions such that if they hold, executing a_i results in a next state that entails $P'(\vec{x})$. To this end, we first identify a predicate $F(\vec{z}) \in \mathcal{F}|\vec{z}:\vec{T_z}$ that can be unified with $P'(\vec{x})$. In PDDL, unification makes the two predicates syntactically identical through variable substitution, but NL-PDDL allows a *more general form of unification based on commonsense entailment*: predicates $F(\vec{z})$ and $P'(\vec{x})$ are unifiable *iff* they have equal arities and a substitution θ exists that induces an injective mapping between their variables such that:

- **(i)** For every pair of corresponding types $t_z \in \vec{T_z}$ and $t_x \in \vec{T_x}$ mapped by θ, $t_x \vdash_{\text{LLM}} t_z$.
- **(ii)** After applying θ, the substituted predicate $F(\vec{z})$ entails the substituted goal predicate $P'(\vec{x})$, i.e., $F(\vec{z})\theta \vdash_{\text{LLM}} P'(\vec{x})\theta$. [2]

A legal typed unification results in a substitution θ that is a mapping of variables to terms (e.g., $\theta = \{x/y\}$, so that $F(x)\theta \equiv F(y)$), which then permits a misaligned regression to proceed. Formally, let $F'_{a_i}(\vec{z}) \in \Phi$ denote the SSA formula for $F'(\vec{z})$ with respect to a_i under the unifier θ:

$$\text{Regr}_{\vdash}\left(P'(\vec{x}), a_i\right) \equiv F'_{a_i}(z)\theta. \tag{3}$$

With $\text{Regr}_{\vdash}$, we can now formally regress a predicate through an action with misaligned effects:

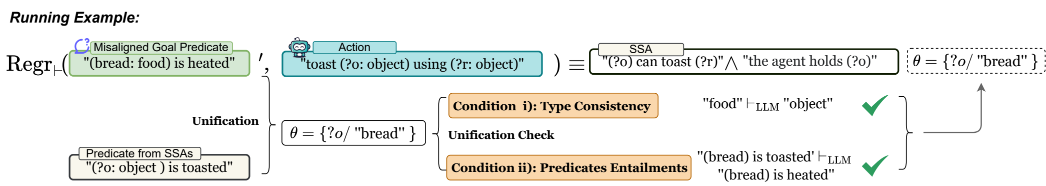

Figure 5: An example of a regression step of a positive predicate with goal–action misalignment.

In the example of Figure 5, to identify the subgoals necessary for the predicate *"(Bread:food) is heated"* to hold after an action is executed, we aim to find a predicate unifiable with it. The predicate *"(?o:object) is toasted"* qualifies, with $\theta = \{$*"?o"/"bread"*$\}$, because **(i)** type consistency holds (*"food"* $\vdash_{\text{LLM}}$ *"object"*), and **(ii)** after substitution, (*"bread is toasted"* $\vdash_{\text{LLM}}$ *"bread is heated"*). This allows us to derive the regressed subgoal condition (given by the SSA) that can achieve *"(bread) is heated"*, which is how subgoal ψ_3 is derived from ψ_2 in the conditional plan example of Figure 2.

Regression of Negative NL predicates. As for negative predicates, we regress a negated predicate $\neg P'(\vec{x})|\vec{x}:\vec{T_x}'$ through an action a_i to derive preconditions that ensure $\neg P'(\vec{x})$ holds after a_i is executed. Thus, we identify a predicate $F(\vec{z}) \in \mathcal{F}|\vec{z}:\vec{T_z}$ that can be unified with $P'(\vec{x})$. Let θ be the unifier inducing an injective mapping between predicate variables such that:

- **(i)** For every pair of corresponding types $t_x \in \vec{T_x}$ and $t_z \in \vec{T_z}$ mapped by θ, $t_x \vdash_{\text{LLM}} t_z$.
- **(ii)** After applying θ, the substituted goal predicate entails the substituted precondition predicate, i.e., $P'(\vec{x})\theta \vdash_{\text{LLM}} F(\vec{z})\theta$.

Critically note that the direction of entailment in check (ii) for negative NL predicates reverses from the case for positive NL predicates. Let $F'_{a_i}(\vec{z}) \in \Phi$ be the SSA formula for $F'(\vec{z})$, with respect to

[2] This relaxation from syntactic equivalence to entailment-based unification allows, for example, unifying *"(?x:food) is on (?y:plate)"* with *"(?y:receptacle) is under (?x:object)"*, as they are both binary and with $\theta = \{?x/?y, ?y/?x\}$, we have *"food"* $\vdash_{\text{LLM}}$ *"object"*, *"plate"* $\vdash_{\text{LLM}}$ *"receptacle"*, and *"(?x) is on (?y)"* $\vdash_{\text{LLM}}$ *"(?y) is under (?x)"*.

a_i. Applying θ to this formula and negating the result gives the regressed condition for $\neg P'(\vec{x})$ with respect to a_i:

$$\mathrm{Regr}_{\vdash}\big(\neg P'(\vec{x}), a_i\big) \equiv \neg F'_{a_i}(\vec{z})\theta. \tag{4}$$

We provide a worked example illustrating the regression of a negative NL predicate in Appendix D.

NL Regression with Multiple Entailments. A predicate $P'(\vec{x})|\vec{x} : \vec{T}$ may be semantically entailed by multiple other predicates. For example, the goal *"(?o) is cooked"* may be entailed by either *"(?o) is toasted"* or *"(?o) is boiled"*. In such cases, we aggregate all entailing predicates.

Let $F^{+}_{\vdash} = \{\, F_1(\vec{x}_1), \ldots, F_m(\vec{x}_m) \mid F_i(\vec{x}_i) \in \mathcal{F} \;\wedge\; F_i(\vec{x}_i)\{\vec{x}_i/\vec{x}\} \vdash_{\mathrm{LLM}} P'(\vec{x}) \,\}$ be the set of all predicates that entail $P'(\vec{x})$. The aggregated regression of $P'(\vec{x})$ with respect to action a_i is:

$$\mathrm{Regr}_{\vdash}\big(P'(\vec{x}), a_i\big) \;\equiv\; \bigvee_{j=1}^{m} F'_{a_i,j}(\vec{x}), \quad \text{where } F_j \in F^{+}_{\vdash}. \tag{5}$$

The same construction applies to negated predicates, with entailment checked in the reverse direction. Let $F^{-}_{\vdash} = \{\, F_1(\vec{x}_1), \ldots, F_m(\vec{x}_m) \mid F_i(\vec{x}_i) \in \mathcal{F} \;\wedge\; P'(\vec{x})\{\vec{x}_i/\vec{x}\} \vdash_{\mathrm{LLM}} F_i(\vec{x}_i) \,\}$ be the set of all predicates entailed by $P'(\vec{x})$. The aggregated regression of $\neg P'(\vec{x})$ with respect to a_i is:

$$\mathrm{Regr}_{\vdash}\big(\neg P'(\vec{x}),\, a_i\big) \;\equiv\; \bigwedge_{j=1}^{m} \neg\, F'_{a_i,j}(\vec{x}), \quad \text{where } F'_j \in F^{-}_{\vdash}. \tag{6}$$

2.1.2 Regression of Formulas

Full first-order formulas can now be regressed using Algorithm 1: the formula is rewritten in Disjunctive Normal Form (DNF), the predicates in each disjunct are regressed following the process in Section 2.1.1, and the results are incrementally combined in DNF. The general regression process in the NL-PDDL framework is summarized in Appendix E.

Algorithm 1 RegressFormula(ψ', $a_i(\vec{y}_i)$, $\mathcal{F}$)

Input: ψ' in DNF ; $a_i(\vec{y}_i)$; $\mathcal{F}$
Output: ψ = $\mathrm{Regr}_{\vdash}(\psi', a_i)$ in DNF
$\psi \leftarrow \varnothing$
for for each disjunct $C \in \psi$ **do**
  $\mathcal{D} \leftarrow \top$
  for each literal $F_j(\vec{x}_j)$ in C **do**
    Construct: $F^{+}_{\vdash}, F^{-}_{\vdash}$
    $F'_{a_i,j}(\vec{x}) \leftarrow \mathrm{Regr}_{\vdash}\big(F_j(\vec{x}),\, a_i\big)$
    $\mathcal{D} \leftarrow \mathcal{D} \wedge F'_{a_i,j}(\vec{x})$
    $\mathcal{D} \leftarrow \mathrm{ConvertToDNF}\big(\mathcal{D}\big)$
  end for
  $\psi \leftarrow \psi \vee \mathcal{D}$
end for
return ψ

3 Experiments

We pose the following research questions:

- RQ1: **Overall Planning Performance.** How does NL-PDDL[3] perform in terms of success rate and computational cost across modalities (e.g., text, images), compared to baselines?
- RQ2: **Reasoning over Language Misalignment.** How well does NL-PDDL handle planning tasks where NL descriptions of the action specifications and the goal are misaligned?
- RQ3: **Effect of Task Complexity.** How does NL-PDDL compare to baselines as task complexity increases, measured by optimal plan length and the number of constraints in the goal?

3.1 Experimental Setup

Open-World Tasks. We conduct open-world planning experiments on *ALFWorld Text* (Shridhar et al., 2021) and *ALFWorld Vision* (cf. Figure 6) to evaluate performance across modalities. Agents must operate under partial observability and incomplete task-relevant knowledge to complete common household tasks in virtual home environments. It supports a rich space of interactions, requiring the agents to reason about object relations and action affordances without hardcoded annotations. Following prior work (Yang et al., 2024; Shridhar et al., 2021; Yao et al., 2023b; Wang et al., 2023), we evaluate NL-PDDL on 135 out-of-distribution tasks with 50 movement budget (cf. Appendix G). We also extend both ALFWorld benchmarks by introducing misaligned language descriptions of the agent's action model and goals (cf. Appendix H).

[3] https://anonymous.4open.science/r/nl_pddl_planner-4C67/README.md

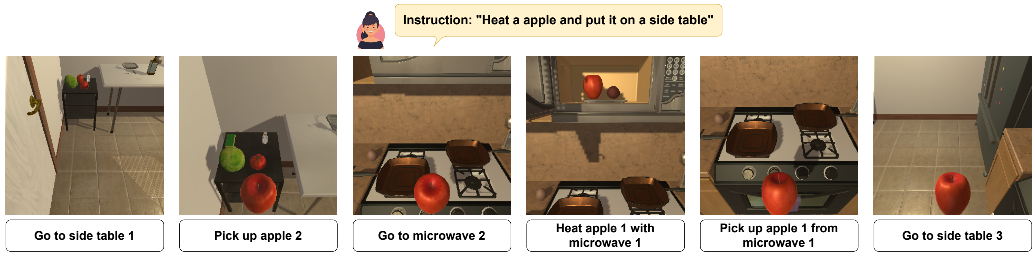

Figure 6: An Example of a Visual ALFWorld Task.

Table 1: **(RQ1)** Performance of different methods in two **open-world** datasets **without misalignment** between goal and model.†Reported performance in the original work.

Open-World Without Misalignment						
Datasets	**Methods**		**# Tok.**	**# Expert Trajectory**	**Fine-Tuning**	**SR (%)**
ALFWorld Text	Direct LLM-based Planner	GPT-4o	1,358,934	0	✗	21%
		Gemini-2.0-Flash	1,176,406	0	✗	16%
		LLaMA-3.1	1,193,640	0	✗	19%
	Reflective LLM-based Planner	ReAct (w examples)	5,507,266	0	✗	81%
		ReAct (w model)	4,242,560	0	✗	34%
		Reflexion-3†	NA	0	✗	83%
		Reflexion-10†	NA	0	✗	91%
		DEPS†	NA	0	✗	76%
	Fine-tuned LM	BUTLER	NA	100,000	✓	26%
	Ours	**NL-PDDL**	443,407	0	✗	**94%**
ALFWorld Vision	Direct VLM-based Planner	GPT-4o	1,823,539	0	✗	8%
		Gemini-2.0-Flash	1,440,025	0	✗	2%
		LLaMA-3.1	1,505,371	0	✗	2%
	Fine-tuned VLM	LLaMA-Adapter†	NA	170,000	✓	13%
		InstructBLIP†	NA	170,000	✓	22%
		EMMA-3†	NA	15,237	✓	37%
		EMMA-10†	NA	15,237	✓	82%
	Ours	**NL-PDDL**	679,148	0	✗	**84%**

Table 2: **(RQ1)** Performance comparison across **closed-world** Blocksworld variants **without misalignment**. Direct LLM-based planners fail to generate valid plans for Mystery and Randomized Blocksworld. NL-PDDL demonstrates consistent and robust performance across all variants.

Closed-World Without Misalignment						
Method	**Blocksworld**		**Mystery Blocksworld**		**Randomized Blocksworld**	
	# Tok.	SR (%)	# Tok.	SR (%)	# Tok.	SR (%)
GPT-4o	982,602	34%	834,890	0%	835,974	0%
Gemini-2.0 Flash	928,024	18%	834,060	1%	835,824	0%
LLaMA 3.1	963,565	44%	841,791	0%	842,622	0%
Fast Downward	N/A	100%	N/A	100%	N/A	100%
NL-PDDL	0	70%	0	70%	0	70%

Closed-World Tasks. *Blocksworld* is a widely used closed-world planning domain in which the task is to rearrange colored blocks, initially stacked or on a table, to achieve a given goal. We adopt a variant introduced in Valmeekam et al. (2022) by converting goal specifications into NL descriptions for NL-based planners. We include two additional variants to probe brittleness to surface form and lexical priors by replacing object and goal names in the NL descriptions with random English words (*Mystery Blocksworld*) and random strings (*Randomized Blocksworld*). Beyond these, we introduce *Misalignment Blocksworld*, where the NL descriptions of action model and goals are contextually meaningful but intentionally misaligned, to assess the commonsense entailment capabilities of planners (cf. Appendix H for details).

Baselines and Evaluation Metrics. We compare NL-PDDL with SOTA baselines in open- and closed-world domains. In the open-world settings (i.e., ALFWorld Text and Vision), we compare with (i) **Direct** LLM/VLM-based planners: GPT-4o (Hurst et al., 2024), Gemini-2.0 Flash (Team et al., 2023), and LLaMA-3.1 (Grattafiori et al., 2024), (ii) **Reflective** LLM-based planners: Re-

Table 3: **(RQ2)** Performance across settings **with misalignment** in open-world ALFWorld (Text/Vision) and closed-world Misalignment Blocksworld. Reported-results-only methods are omitted since prior works did not consider misalignment.

Open-World With Misalignment							
Category	**Method**	**ALFWorld Text**		**ALFWorld Vision**		**Blocksworld**	
		# Tok.	SR (%)	# Tok.	SR (%)	# Tok.	SR (%)
Direct LLM	GPT-4o	1,440,470	17% (↓5%)	1,811,404	7% (↓1%)	937,905	27% (↓7%)
	Gemini-2.0 Flash	1,251,533	15% (↓1%)	1,262,016	5% (↑3%)	939,950	23% (↑5%)
	LLaMA-3.1	1,310,664	15% (↓4%)	1,480,392	2% (-0%)	950,490	42% (↓3%)
ReAct	w/ examples	5,215,612	79% (↓12%)	N/A	N/A	N/A	N/A
	w/ model	4,428,914	23% (↓11%)	N/A	N/A	N/A	N/A
Classical Planner	Fast Downward	N/A	N/A	N/A	N/A	0	0%
Ours	**NL-PDDL**	501,049	**91%** (↓3%)	745,365	**80%** (↓4%)	11,656	**70%**

Act (Yao et al., 2023b), Reflexion (Shinn et al., 2023), and DEPS (Wang et al., 2023), and (iii) **Fine-tuned** LLM/VLMs: BUTLER (Shridhar et al., 2021), LLaMA-Adapter (Gao et al., 2023), InstructBLIP (Dai et al., 2023), and EMMA (Yang et al., 2024), all trained on ALFWorld expert demonsrations. In the closed-world setting, we compare against Direct LLM-based planners and the classical Fast Downward (Helmert, 2006) planner, which serves as an upper bound on planning performance. The detailed implementation of NL-PDDL on the ALFWorld and Blocksworld benchmarks is included in Appendix I. Methods are evaluated under a feasible maximum runtime. For evaluation, we consider two aspects: task *Success Rate* (SR), and associated cost, as captured by the *# Tokens* Usage (# Tok.) in LLM.

4 Results and Discussion

RQ1. In open-world tasks (cf. Table 1), NL-PDDL achieves superior SR on both ALFWorld Text and Vision compared to all baselines. On ALFWorld Text, NL-PDDL achieves a 94% SR with only 443K tokens, whereas ReAct consumes over ten times more tokens while reaching a lower SR. Although Reflexion-10 achieves performance comparable to NL-PDDL, it depends on repeating ten trials of the same task, which is infeasible for many embodied AI applications. Impressively, NL-PDDL even outperforms all fine-tuned VLMs that rely on thousands of expert trajectories in visual ALFWorld. Our results highlight its generalizability across different modalities for open-world tasks. In closed-world tasks (cf. Table 2), NL-PDDL demonstrates robust performance with a consistent 70% SR at 0 tokens. In contrast, direct LLM-based planners entirely fail to solve the Mystery and Randomized variants, performing with less than 1% SR despite substantial token usage. Fast Downward, a closed-world planner, achieves a 100% success rate on the three Blocksworld domains. However, it is brittle under goal–action misalignment, as we demonstrate in RQ3.

RQ2. In Table 3, we evaluate the performance of different methods on open- and closed-world tasks with misalignment between the NL descriptions of action models and goals. NL-PDDL demonstrates clear robustness in the face of such misalignments. In open-world tasks, NL-PDDL maintains a 91% SR on ALFWorld Text and 80% on ALFWorld Vision. By contrast, ReAct, whether explicitly prompted to focus on action models or supported with few-shot examples, suffers notable degradation, underscoring its sensitivity to misalignment. The closed-world Misalignment Blocksworld variant highlights the gap: Fast Downward, though highly effective in purely symbolic and aligned settings, does not perform commonsense entailment and fails under goal—action schema misalignment; in contrast, NL-PDDL maintains performance comparable to other Blocksworld variants, evidencing robust misalignment handling.

RQ3. We evaluate planner performance relative to optimal plan depth. NL-PDDL achieves perfect success up to depth 6, with performance declining to 84% at depth 8 and fails beyond depth 10 under the runtime limit. By contrast, LLM-based planners exhibit consistent deterioration across all depths. NL-PDDL's decline stems from scalability limitations under the predefined maximum runtime, while LLM-based planners fail due to their inability to reason over an extended horizon. The Fast Downward planner maintains perfect performance when the action model and goals are fully aligned, but fails on the Misalignment Blocksworld where commonsense entailment reasoning is required. In open-world tasks where optimal depth is inconsistent, we measure NL-PDDL's

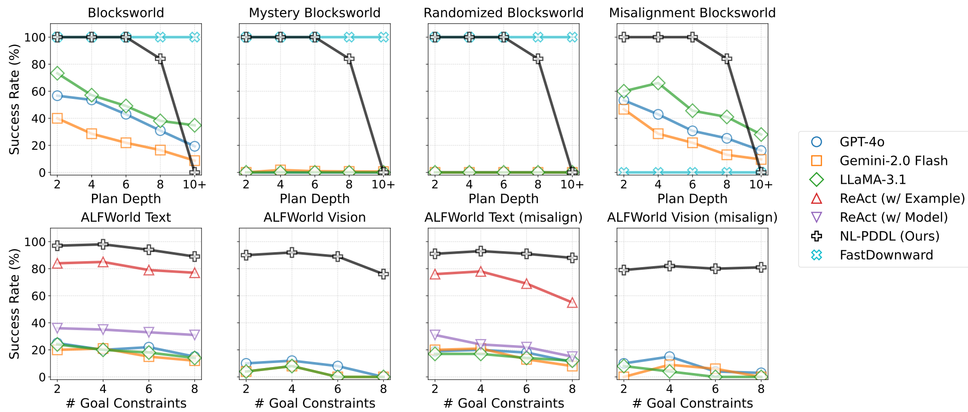

Figure 7: **(RQ3)** SR by goal complexity and optimal plan depth in ALFWorld and Blocksworld across planners

performance against goal complexity. NL-PDDL remains stable, showing only a 3% average drop across ALFWorld Text and Vision, while LLM/VLM-based planners degrade much more sharply, underscoring their sensitivity to increasing goal complexity. Together, these results demonstrate the robustness of NL-PDDL with respect to goal complexity across open- and closed-world tasks.

5 Related Work

Eliciting Stronger Reasoning from LLMs: As LLMs scale, they remain prone to hallucinations and logical fallacies (Tonmoy et al., 2024; Zhang et al., 2025) resulting in new prompting strategies to mitigate these concerns. Step-by-step methods (Wei et al., 2022; Kojima et al., 2022; Yao et al., 2023a) encourage incremental reasoning by breaking problems into sub-steps or branching search trajectories. Reflective methods such as ReAct (Yao et al., 2023b) and Reflexion (Shinn et al., 2023) add self-reflection, critiquing, and revision. Yet these methods often fail on tasks with several logical constraints or long-horizon dependencies (Valmeekam et al., 2023b; 2022; Song et al., 2025); are sensitive to prompt design and few-shot examples, iteration dependent (Pan et al., 2023); and remain unverifiable since the LLM acts as a black box without guarantees of soundness (Shanahan, 2024).

Planning with LLMs and Symbolic Planners: A separate body of work integrates LLMs with classical planners. One approach treats LLMs as planners that generate action sequences from symbolic task descriptions (Silver et al., 2022; Song et al., 2023; Silver et al., 2024). Another uses LLMs as *translators*, converting NL problems and domains into structured formats for symbolic solvers (Guan et al., 2023; Kambhampati et al., 2024; Oswald et al., 2024; Yang et al., 2023). Both approaches face limitations: LLM-as-planner approaches are unreliable due to reasoning errors and hallucinations, while translator-based methods often introduce faulty predicates, actions, or logical mappings of NL descriptions that cause downstream planning failures. Translation also imposes an expressivity bottleneck, restricting applicability to problems encodable in symbolic form. Finally, most assume closed-world settings, making them inappropriate for open-world planning.

Our Point of Departure: We introduce a novel NL-PDDL hybrid natural language variation of PDDL to facilitate the use of powerful commonsense (V)LLM reasoning during planning, and blend it with symbolic *lifted* regression over NL-PDDL to address open-world, long-horizon planning.

We further discuss NL-to-PDDL translation approaches and clarify their distinctions from NL-PDDL in Appendix J.

6 Conclusion

We proposed NL-PDDL, a framework for open-world planning that extends symbolic PDDL to support NL, improving accessibility for non-experts. NL-PDDL facilitates the integration of LLM commonsense knowledge into open-world regression planning to address goal–action misalignment, while retaining soundness and verifiability. Empirically, we showed that lifted regression planning in NL-PDDL achieves higher plan-success rates in contrast to existing strong baselines, remains robust as plan horizons increase, and generalizes well across both text and vision modalities.

REFERENCES

Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins Sri, Anthony Barrett, Dave Christianson, et al. Pddl—the planning domain definition language. *Technical Report, Tech. Rep.*, 1998.

Mohamed Aghzal, Erion Plaku, Gregory J Stein, and Ziyu Yao. A survey on large language models for automated planning. *arXiv preprint arXiv:2502.12435*, 2025.

Pierre Carbonnelle. pydatalog: Logic programming in python. `https://sites.google.com/site/pydatalog/home` or `https://github.com/pcarbonn/pyDatalog`, 2024.

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale N Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning. *Advances in neural information processing systems*, 36:49250–49267, 2023.

Peng Gao, Jiaming Han, Renrui Zhang, Ziyi Lin, Shijie Geng, Aojun Zhou, Wei Zhang, Pan Lu, Conghui He, Xiangyu Yue, et al. Llama-adapter v2: Parameter-efficient visual instruction model. *arXiv preprint arXiv:2304.15010*, 2023.

Alfonso Emilio Gerevini. An introduction to the planning domain definition language (pddl): Book review. *Artificial Intelligence*, 280:103221, 2020.

Kai Goebel and Patrik Zips. Can llm-reasoning models replace classical planning? a benchmark study. *arXiv preprint arXiv:2507.23589*, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094, 2023.

Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, Christian Muise, Ronald Brachman, Francesca Rossi, and Peter Stone. *An introduction to the planning domain definition language*, volume 13. Springer, 2019.

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

Yongfeng Huo, Jing Tang, Yinghui Pan, Yifeng Zeng, and Langcai Cao. Learning a planning domain model from natural language process manuals. *IEEE Access*, 8:143219–143232, 2020.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. Llms can't plan, but can help planning in llm-modulo frameworks. *arXiv preprint arXiv:2402.01817*, 2024.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

Xiangrui Kong, Wenxiao Zhang, Jin Hong, and Thomas Braunl. Embodied ai in mobile robots: Coverage path planning with large language models. *arXiv preprint arXiv:2407.02220*, 2024.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Erran Li Li, Ruohan Zhang, et al. Embodied agent interface: Benchmarking llms for embodied decision making. *Advances in Neural Information Processing Systems*, 37:100428–100534, 2024.

Shivam Miglani and Neil Yorke-Smith. Nltopddl: one-shot learning of pddl models from natural language process manuals. In *Proc. of the ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS). ICAPS*, 2020.

Christian Muise, Florian Pommerening, Jendrik Seipp, and Michael Katz. Planutils: Bringing planning to the masses. In *ICAPS 2022 System Demonstrations*, 2022.

James Oswald, Kavitha Srinivas, Harsha Kokel, Junkyu Lee, Michael Katz, and Shirin Sohrabi. Large language models as planning domain generators. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pp. 423–431, 2024.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295*, 2023.

Raymond Reiter. Equality and domain closure in first-order databases. *Journal of the ACM (JACM)*, 27(2):235–249, 1980.

Raymond Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy*, 50:359–380, 1991.

Murray Shanahan. Talking about large language models. *Communications of the ACM*, 67(2):68–79, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *arXiv preprint arXiv:2506.06941*, 2025.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Tom Silver, Varun Hariprasad, Reece S Shuttleworth, Nishanth Kumar, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Pddl planning with pretrained large language models. In *NeurIPS 2022 foundation models for decision making workshop*, 2022.

Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B Tenenbaum, Leslie Kaelbling, and Michael Katz. Generalized planning in pddl domains with pretrained large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 20256–20264, 2024.

Pavel Smirnov, Frank Joublin, Antonello Ceravola, and Michael Gienger. Generating consistent pddl domains with large language models. *arXiv preprint arXiv:2404.07751*, 2024.

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

Peiyang Song, Pengrui Han, and Noah Goodman. A survey on large language model reasoning failures. In *2nd AI for Math Workshop@ ICML 2025*, 2025.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

SM Tonmoy, SM Zaman, Vinija Jain, Anku Rani, Vipula Rawte, Aman Chadha, and Amitava Das. A comprehensive survey of hallucination mitigation techniques in large language models. *arXiv preprint arXiv:2401.01313*, 6, 2024.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change, 2023a. URL `https://arxiv.org/abs/2206.10498`.

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

Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. Llms still can't plan; can lrms? a preliminary evaluation of openai's o1 on planbench. *arXiv preprint arXiv:2409.13373*, 2024.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

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

Yijun Yang, Tianyi Zhou, Kanxue Li, Dapeng Tao, Lusong Li, Li Shen, Xiaodong He, Jing Jiang, and Yuhui Shi. Embodied multi-modal agent trained by an llm from a parallel textworld. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 26275–26285, 2024.

Zhun Yang, Adam Ishay, and Joohyung Lee. Coupling large language models with logic programming for robust and general reasoning from text. *arXiv preprint arXiv:2307.07696*, 2023.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. Siren's song in the ai ocean: A survey on hallucination in large language models. *Computational Linguistics*, pp. 1–46, 2025.

A Ethics, Reproducibility, LLM Usage Statement

A.1 Ethics Statement

We adhere to the ICLR Code of Ethics. This work does not involve human or animal subjects, and no identifiable personal information was used in developing this paper. Therefore, there are no privacy or security concerns. We are committed to maintaining integrity and transparency throughout the research process.

A.2 Reproducibility Statement

We have made every effort to ensure that the methodology presented in this work is reproducible. All code used in this study has been released in an anonymous repository to support reproducibility and verifiability. Full descriptions of the methods and experimental setup are provided in both the main text and the appendix to facilitate transparent evaluation and replication of our results.

A.3 The Use of Large Language Models (LLMs)

Large Language Models (LLMs) were used to aid or polish the writing of this paper. Specifically, we employed LLMs to improve the readability and clarity of the manuscript.

B Fundamentals of Regression Planning in PDDL

In this section, we review PDDL and its representation of planning problems, then introduce the first-order regression approach to planning. The corresponding operations for our proposed NL-PDDL are presented in Section 2.1 of the main paper.

B.1 PDDL Planning Representation.

A planning problem is defined as the tuple $\mathcal{P} = \langle \mathcal{F},\ \mathcal{O},\ \mathcal{A},\ s_0,\ G \rangle$, where:

- $\mathcal{F}$ is the set of first-order predicates that describe properties of objects or relations between them, whose truth values may change as actions are applied, such as `boiled(x)`, $\texttt{can\_toast}(x, y)$, etc.
- $\mathcal{O}$ is the set of objects in the domain, which serve as the constants over which predicates and actions are instantiated, (e.g., *potato*, *plate*, etc.).
- $\mathcal{A}$ is the set of actions $a_i(\vec{y})$, i.e., parameterized first-order operators. Each action schema is defined by:
  - *Preconditions:* Denoted by $a_i(\vec{y_i}).\texttt{pre} \subset \mathcal{F}$ is the set of predicates that must hold for the action to be applicable.
  - *Add effects:* Denoted by $a_i(\vec{y_i}).\texttt{add} \subset \mathcal{F}$ is the set of predicates that become true once the action is executed.
  - *Delete effects:* Denoted by $a_i(\vec{y_i}).\texttt{del}$ is the set of predicates that are no longer true after the action is executed.

  For instance, the action `pickup(?o)` has the preconditions `hand_empty` and `near(?o)`, the add effect `possess(?o)`, and the delete effect `hand_empty`.
- s_0 is the initial state, given as a set of ground predicates that hold true before any action is executed.
- G is the goal condition, specified as a set of predicates that must be satisfied in a terminal state.

In classical planning, problems are often specified in the *Planning Domain Definition Language (PDDL)*, which is divided into two parts: the *domain* and the *problem instance*. The domain defines the agent's *action model* together with the set of predicates $\mathcal{F}$ and the set of action schemas $\mathcal{A}$. The *problem instance* specifies a finite object set $\mathcal{O}$, an initial state s_0 and G given as a set of ground predicates.

The objective of solving a planning problem is to find a *conditional plan*, i.e., a finite sequence of actions $\pi = \langle a^{(1)}, \ldots, a^{(n)} \rangle$, such that, starting from s_0, executing each action in π sequentially results in a state s_n that satisfies the goal G. Here, the superscript (i) in $a^{(i)}$ indicates that a is the action executed at the i-th step of the plan, i.e., it represents the horizon index of the action within the sequence.

Example. We represent the planning problem as

$$\mathcal{P} = \langle \mathcal{F},\ \mathcal{O},\ \mathcal{A},\ s_0,\ G \rangle.$$

predicates $\mathcal{F}$.

$$\mathcal{F} = \{\texttt{hand\_empty},\ \texttt{near}(x), \texttt{possess}(x), \texttt{on}(x,y), \texttt{isHot}(x)\ \texttt{can\_bake}(x,y), \texttt{full\_of\_water}(y), \texttt{baked}(x),\ \texttt{boiled}(x)\}.$$

Objects $\mathcal{O}$.

$$\mathcal{O} = \{\ \textit{bread},\ \textit{plate},\ \textit{toaster},\ \textit{pot}\ \}.$$

Actions $\mathcal{A}$.

pickup(?o) :

pre: {hand_empty , near(?o)}
add: {possess(?o)}
del: {hand_empty}

puton(?o, ?r):

pre: {possess(?o), near(?r)}
add: {hand_empty, on(?o,?r)}
del: {possess(?o)}

toast(?o, ?r):

pre: { can_toast(?o,?r), possess(?o)}
add: {isHot(?o)}
del: $\varnothing$

Initial state s_0.

s_0 = {hand_empty, near(potato), near(plate), near(oven),near(pot), boiling_device(pot), can_toast(bread,toaster),full_of_water(pot)}.

Goal G.

G = { heated(bread), on(bread,plate) }.

B.2 First Order Regression

First-order Regression addresses open-world planning by lifting object representations into variables and reasoning backward from goals. First-order regression allows the agent to generate lifted plans without requiring prior knowledge of the full object set $\mathcal{O}$ or a complete initial state s_0. In this approach, beginning from the goal, the objective is to iteratively regress the target formula through applicable actions, in order to compute the subgoals that must hold in earlier states. By repeating this process, the agent constructs a sequence of actions that achieves the original goal while remaining fully symbolic and lifted, enabling reasoning over unknown or partially observed objects.

In this section, we introduce the procedure for translating a PDDL problem into a first-order representation and explain how regression is performed using the Successor State Axioms (SSAs).

B.2.1 PDDL to First-Order Logic Domain Theory

To axiomatize a PDDL domain theory, we begin by defining positive and negative effect axioms, which specify how predicates change as a result of actions.

- **Positive effect axioms**: state which actions $a \in \mathcal{A}$ can explicitly make each predicate $F(\vec{x})$ true.
- **Negative effect axioms** state which actions $a \in \mathcal{A}$ can explicitly make the predicate $F(\vec{x})$ false.

These axioms are essential for accurately regressing a goal through an action: to determine the conditions that must have held before the action, one needs to know exactly how each action modifies the state. To obtain these axioms, we make the following assumptions.

- No new variables outside of action parameter are introduced in preconditions and effects.
- No quantifiers are used in the action's preconditions and effects.
- All unquantified variables are implicitly universally quantified.

We assume positive and negative effect axioms can be specified by considering all of the ways in which each action can affect each predicate. Let $F(\vec{y_F})$ be a predicate in the current state, $\gamma^+_{F,a_i}(\vec{y}, s)$ is a first-order formula such that, if it holds in the current state s, then $F'(\vec{y_F})$ holds after executing $a_i(\vec{y})$. Similarly, let $\gamma^-_{F,a_i}(\vec{y}, s)$ be a first-order formula such that, if it holds in the current state s, then $\neg F'(\vec{y_F})$ holds after executing $a_i(\vec{y})$. We write $\gamma^+_{F,a_i}(\vec{y})$ and $\gamma^-_{F,a_i}(\vec{y})$ as shorthands for the state-dependent conditions $\gamma^+_{F,a_i}(\vec{y}, s)$ and $\gamma^-_{F,a_i}(\vec{y}, s)$ in the current state, respectively, and define the normal form of effect axioms as following:

$$\forall \vec{y} : \vec{T} \left[\gamma^+_{F,a_i}(\vec{y}) \;\Rightarrow\; F'_{a_i}(\vec{y}_F)\right], \qquad \forall \vec{y} : \vec{T} \left[\gamma^-_{F,a_i}(\vec{y}) \;\Rightarrow\; \neg F'_{a_i}(\vec{y}_F)\right]. \tag{7}$$

Here, $\vec{y} : \vec{T}$ indicates that each variable in $\vec{y}$ is assigned its type by a corresponding element in $\vec{T}$. Given a PDDL problem $\mathcal{P}$ with an action set $\mathcal{A}$, we can derive effect axioms in the aforementioned normal form, action by action. Consider an action $a(\vec{y}) \in \mathcal{A}$, and let $F(\vec{y}_F)$ be a predicate in its positive effect set, i.e., $F(\vec{y}_F) \in a(\vec{y}).\texttt{add}$. The predicate parameters $\vec{y}_F$ correspond to a permutation of a subset of the action parameters $\vec{y}$. We use $\vec{y}_{\setminus F}$ to denote the remaining action parameters that do not appear in $\vec{y}_F$ which are existentially quantified. With this setup, we construct the following implication:

$$\text{for each } F(\vec{y}_F) \in a_i(\vec{y}).\texttt{add}:\; \forall \vec{y} : \vec{T}, a\; \underbrace{\Big[a = a_i(\vec{y}) \;\wedge \bigwedge_{\text{Pre}_j(\vec{y_j}) \in a_i(\vec{y}).\texttt{pre}} \text{Pre}_j(\vec{y_j})\Big]}_{\gamma^+_{F,a_i}(\vec{y})} \Rightarrow F'_{a_i}(\vec{y}_F). \tag{8}$$

Similarly, for the predicates in the delete effects of the action, we have:

$$\text{for each } F(\vec{y}_F) \in a_i(\vec{y}).\texttt{del}:\; \forall \vec{y} : \vec{T}, a\; \underbrace{\Big[a = a_i(\vec{y}) \;\wedge \bigwedge_{\text{Pre}_j(\vec{y_j}) \in a_i(\vec{y}).\texttt{pre}} \text{Pre}_j(\vec{y_j})\Big]}_{\gamma^-_{F,a_i}(\vec{y})} \Rightarrow \neg F'_{a_i}(\vec{y}_F). \tag{9}$$

We can combine multiple predicates with the same name via a disjunction. The detailed methodology is outlined in Appendix C.

Positive effect axiom. For `isHot(?o)` with respect to the action `toast(?o, ?r)`:

$$\forall \texttt{?o:object}, \texttt{?r:object}, a \left(\underbrace{a = \texttt{toast(?o,?r)} \wedge \texttt{can\_toast(?o,?r)} \wedge \texttt{possess(?o)}}_{\gamma^+_{\texttt{isHot},\,\texttt{toast}}(\texttt{?o,?r})} \Rightarrow \texttt{isHot'(?o)} \right).$$

Negative effect axiom. For `possess(?o)` with respect to the action `puton(?o, ?r)`:

$$\forall \texttt{?o:object}, \texttt{?r:receptacle}, a \left(\underbrace{a = \texttt{puton(?o,?r)} \wedge \texttt{near(?r)} \wedge \texttt{possess(?o)}}_{\gamma^-_{\texttt{possess},\,\texttt{puton}}(\texttt{?o,?r})} \Rightarrow \neg\texttt{possess'(?o)} \right).$$

Succesor State Axioms When a PDDL problem is transformed into the above normal form, Reiter (1991) showed that, under the assumptions of the *Unique Names Axioms* (Reiter, 1980) and the *Explanation Closure Axioms* (Reiter, 1991), we can construct *Successor State Axioms* (SSAs) that capture how a predicate may change or persist as the agent interacts with the environment. Let $F'_{a_i}(\vec{y_F})$ denote the value of a predicate $F(\vec{y_F})$ in the next state after executing action $a_i(\vec{y})$. The SSA for $F(\vec{y_F})$ is defined as:

$$\forall \vec{y} : \vec{T}\ [F'_{a_i}(\vec{y_F}) \equiv \gamma^+_{F,a_i}(\vec{y})\ \vee\ \big(F(\vec{y_F}) \wedge \neg\gamma^-_{F,a_i}(\vec{y})\big)]. \tag{10}$$

Intuitively, SSAs state that a predicate $F'(\vec{y_F})$ in the next state can be true either because it is made true by an action $a_i(\vec{y})$, as specified by $\gamma^+_{F,a_i}(\vec{y})$, or because it was already true in the previous state, $F(\vec{y_F})$, and the action does not make it false, i.e., $\neg\gamma^-_{F,a_i}(\vec{y})$.

$$\forall\, \texttt{?o:object},\ \texttt{?r:receptacle},\ a$$
$$\Big[\texttt{possess'(?o)} \equiv \underbrace{\big(a = \texttt{pickup(?o)} \wedge \texttt{hand\_empty} \wedge \texttt{near(?o)}\big)}_{\gamma^+_{\texttt{possess, pickup}}(\texttt{?o})}$$
$$\vee\ \Big(\texttt{possess(?o)} \wedge \neg \underbrace{\big(a = \texttt{puton(?o,?r)} \wedge \texttt{near(?r)}\big)}_{\gamma^-_{\texttt{possess, puton}}(\texttt{?o,?r})}\Big)\Big].$$

First-Order Regression Let ψ' denote a first-order state description that holds *after* executing an action a_i. The regression operator $Regr(\psi', a_i)$ "backprojects" ψ' to compute a logical formula ψ that must hold *before* the execution of a_i. Fortunately, SSAs provide us a logically equivalent pre-action condition for each predicate $F'_{a_i}(\vec{x}) \in \psi'$ with respect to an action $a_i \in \mathcal{A}$. Regression of the entire formula ψ' is performed by recursively replacing each post-action predicate $F'(\vec{x})$ with its corresponding precondition formula $F'_{a_i}(\vec{x})$, as defined by the appropriate SSA.

$$\textit{Regr}(F'(\vec{x}), a_i) \equiv F'_{a_i}(\vec{x}) \tag{11}$$

$$\textit{Regr}\big(\texttt{isHot(?o)},\texttt{toast(?o, ?r)}\big) \equiv \texttt{can\_toast(?o,?r)} \wedge \texttt{possess(?o)}$$

SSAs lay the foundation of lifted regression planning, since using SSAs, we are able to replace a predicate in a post-action state with the subgoals that are required to hold before the action is executed to achieve the predicate. We provide examples of this functionality in Section 2.1.

C Combining Effect Axioms

Combining Effect Axioms(Appendix) For predicates that appear in multiple action effects, we need to combine them into a single effect axiom. For instance, we have two antecedent condition formulae $C_1(\vec{x}_1, \vec{y}_1)$ and $C_2(\vec{x}_2, \vec{y}_2)$ where:

$$\forall \vec{x}_1 : \vec{T_{x_1}}, \vec{y}_1 : \vec{T_{y_1}}\ [C_1(\vec{x}_1, \vec{y}_1) \Rightarrow F(\vec{x}_1)], \tag{12}$$

$$\forall \vec{x}_2 : \vec{T_{x_2}}, \vec{y}_2 : \vec{T_{y_2}}\ [C_2(\vec{x}_2, \vec{y}_2) \Rightarrow F(\vec{x}_2)]. \tag{13}$$

let $\theta = \{\vec{x}_2 \mapsto \vec{x}_1, y_2 \mapsto \vec{y}_1\}$ be the most general unifier (MGU) of C_1, and C_2, i.e., the substitution that unifies the variables of C_2 with those of the C_1 without introducing unnecessary restrictions. We then apply θ to $C_2(\vec{x}_2, \vec{y}_2)$, i.e., $\mathrm{SUBST}(\theta, C_2(\vec{x}_2, \vec{y}_2)) = C_2(\vec{x}_1, \vec{y}_1)$, and form the following implication:

$$\forall \vec{x_1} : \vec{T_{x_1}}, \vec{y_1} : \vec{T_{y_1}}\ [C_1(\vec{x}_1, \vec{y}_1) \vee C_2(\vec{x}_1, \vec{y}_1)] \Rightarrow \gamma^+_{F,a}(\vec{x}_1, \vec{y}_1), \tag{14}$$

where $\mathrm{SUBST}(\theta, p)$ denotes the formula obtained by applying substitution θ to p.

$$
\begin{aligned}
&\forall a : \mathcal{A},\ \texttt{?o} : \texttt{object},\ \texttt{?r} : \texttt{object}\ \Big[\\
&\quad \big(a = \texttt{bake(?o, ?r)} \wedge \texttt{baking\_device(?r)} \wedge \texttt{can\_bake(?o, ?r)} \wedge \texttt{possess(?o)}\big)\\
&\quad \vee\\
&\quad \big(a = \texttt{boil(?o, ?r)} \wedge \texttt{boiling\_device(?r)} \wedge \texttt{full\_of\_water(?r)} \wedge \texttt{possess(?o)}\big)\Big]\\
&\qquad \Rightarrow \texttt{isHot'(?o)}
\end{aligned}
$$

D EXAMPLE FOR REGRESSION OF NEGATIVE NL PREDICATES

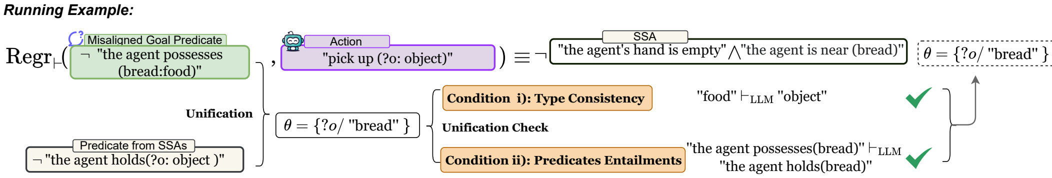

Figure 8: An example of a regression step of a negative predicate with goal–action misalignment.

The process of regressing a negative NL predicate is summarized in Figure 8. To identify the subgoals necessary for the predicate ¬*"the agent possesses(bread:food)"* to hold after an action is executed, we aim to find a predicate that is unifiable with it. The predicate ¬*"the agent holds(?o:object)"* qualifies for this, under the substitution $\theta = \{$*"?o"/"bread"*$\}$, because **(i)** type consistency condition holds (*"food"* $\vdash_{\text{LLM}}$ *"object"*), and **(ii)** once the substitution is applied, (*"the agent possesses (bread)"* $\vdash_{\text{LLM}}$ *"the agent holds (bread)"*). Note that the direction of the entailment in condition **(ii)** is the opposite of the case of positive predicates. The post-action predicate ¬*"the agent possesses(bread:food)"* can thus be unified with the post-action predicate ¬*"the agent holds(?o:object)"*, which is in turn equivalent to its SSA with the action *"pick up (?o)"*, which yields the new subgoal.

E NL-REGRESSION ALGORITHM

Algorithm 2 presents the regression process in the NL-PDDL framework. Starting from G, we standardize actions and regress the current subgoal ψ' through them. The result is simplified and if valid, the action is appended to the plan, forming $(\psi, \vec{a})$. This process is repeated to depth h, yielding the final plan Π which consists of all such pairs $(\psi, \vec{a})$ where ψ is logically equivalent to the original goal G.

Algorithm 2 NL-REGRESSION ALGORITHM

Input: $T = \langle G,\ \mathcal{A},\ h,\ \mathcal{F} \rangle$
Output: $\Pi \subseteq \{\, (\psi, \vec{a}) \,\}$ with each ψ in DNF
$\Phi \;=\; \left\{\, F'_{a_i}(\vec{x}) \;\middle|\; F(\vec{x}) \in \mathcal{F},\ a_i(\vec{y}) \in \mathcal{A} \,\right\}.$
$Frontier \leftarrow \{\, (G,\ [\,],\ 0) \,\}$
$\Pi \leftarrow \{\, (G,\ [\,],\ \varnothing) \,\}$
while $Frontier \neq \varnothing$ **do**
  Extract $(\psi',\ \vec{a},\ d)$ from $Frontier$
  if $d \geqslant h$ **then**
    continue
  end if
  for each $a_i(\vec{y}) \in \mathcal{A}$ **do**
    $a_i(\vec{y}) \leftarrow \text{STANDARDIZE}(a_i(\vec{y}))$ //See Appendix F.1
    $\psi \leftarrow \text{REGRESSFORMULA}\big(\psi',\ a_i(\vec{y}),\ \mathcal{F}\big)$
    if ψ is evaluated to False **then**
      continue
    end if
    $\psi \leftarrow \text{SIMPLIFY}(\psi)$ *//See Appendix F.2*
    $\vec{a}' \leftarrow \vec{a} \,\|\, [a_i(\vec{y})]$
    $Frontier \leftarrow Frontier \cup \{\, (\psi,\ \vec{a}',\ d+1) \,\}$
    $\Pi \leftarrow \Pi \cup \{\, (\psi,\ \vec{a}') \,\}$
  end for
end while
return Π

F OPERATIONS IN FIRST-ORDER REGRESSION

F.1 STANDARDIZATION

Let φ be a first-order formula. The *standardization* of φ, denoted $\text{STANDARDIZE}(\varphi)$, is obtained by renaming the bound variables in φ with fresh variables so that no two distinct quantifiers in φ bind the same variable symbol.

Formally, if $x_1, \ldots, x_k$ are the bound variables in φ, then

$$\text{STANDARDIZE}(\varphi) \;=\; \varphi[\rho(x_1), \ldots, \rho(x_k)],$$

where ρ is a bijective renaming function mapping each x_i to a fresh variable x'_i such that $x'_i \notin \text{Var}(\varphi)$ for all i. Here, $\text{Var}(\varphi)$ denotes the set of variables occurring in the formula φ.

F.2 SIMPLIFICATIONS

Regression typically generates a large, expanded DNF formula. To keep the resulting plan compact and interpretable, we apply a set of simplification rules that maintain logical equivalence while eliminating redundancy. These simplifications enable the planner to express subgoals clearly and prevent repeated formulas from arising during regression.

F.2.1 EQUALITY-BASED QUANTIFIER ELIMINATION

We eliminate quantified variables that are equal to an object by replacing them with the object, i.e.,

$$\exists x : T, [x = x^* \wedge F(x)] \equiv F(x^*)$$

where x^* is an object. This transformation preserves logical equivalence while eliminating redundant quantification. It is often applied during regression, where it is triggered by the need to unify variables between predicates and action parameters. This rule is especially valuable when the regressed formula includes equality constraints that enforce correspondences between variables.

F.2.2 CONTRADICTION EVALUATION

Contradictions often arise within regressed formulas and can be simplified to $\perp$ (false). We consider two main cases:

- **Conjunctive Formulas.** A conjunction that contains mutually exclusive literals or unsatisfiable equality conditions simplifies to false. For example:

$$F(x) \land \neg F(x) \quad \to \quad \bot,$$
$$x = y \land x \neq y \quad \to \quad \bot.$$

  These situations typically occur when inconsistent conditions are introduced through regression or substitution.

  Disjunctive Formulas. In disjunctions, the false literal $\bot$ can be eliminated since it does not affect the overall satisfiability:

$$\bot \lor F(x) \lor \dots \quad \equiv \quad F(x) \lor \dots$$

F.2.3 No No-Op Assumption.

We assume that the domain does not contain *no-op actions*. A no-op action is an action that leaves the state unchanged, so we assume that every action makes a meaningful change to the state. Hence, any disjunct in the regressed formula that is logically identical to the original goal can be safely eliminated.

$$\text{Regr}(G(x)) \quad \equiv \quad \bigvee_i \Psi_i^{DNF} \;\lor\; G(x) \quad \equiv \quad \bigvee_i \Psi_i'^{DNF}$$

This guarantees that plans are constructed only from informative regressions.

F.2.4 DNF Subsumption.

Let ϕ be a DNF formula:

$$\phi = \bigvee_{i=1}^{n} C_i,$$

in which each clause C_i is a conjunction of literals. We say clause C_i *subsumes* another clause C_j if

$$C_i \Rightarrow C_j.$$

In this simplification technique, we eliminate any clause C_j for which there exists another clause C_i such that $C_i \Rightarrow C_j$. Formally, if

$$\exists i \neq j, \quad \text{such that} \quad C_i \Rightarrow C_j,$$

the formula

$$\phi = \bigvee_{i=1}^{n} C_i$$

is simplified to

$$\phi' = \bigvee_i C_i \quad \text{such that} \quad \forall j, k : (j \neq k) \Rightarrow \neg(C_j \Rightarrow C_k)$$

For example, with DNF subsumption, we have

$$[F_1(\vec{x_1}) \land F_2(\vec{x_2})] \land [F_1(\vec{x_1}) \land F_2(\vec{x_2}) \land F_3(\vec{x_3})] \equiv [F_1(\vec{x_1}) \land F_2(\vec{x_2})].$$

This simplification allows the planner to just maintain the most general subgoal formulas while preserving correctness.

F.2.5 Duplicate Detection.

To prevent inefficiency during plan generation, a process called duplicate detection ensures the planner doesn't repeatedly visit the same subgoal. Exploring logically equivalent but distinct branches would lead to wasted computation and overly complex policies.

A regressed formula is marked as a duplicate if it is structurally identical to one already seen, ignoring differences in how variables are named or how the conjuncts are ordered.

To enable this duplicate detection, every conjunctive formula is standardized by performing the following steps:

- *Canonicalizing order*: Sorting all predicates and their arguments into a fixed, canonical sequence.
- *Consistent variable mapping*: Assigning variables consistently across the formula (e.g., uniformly replacing x, y, z with v_1, v_2, v_3 in corresponding positions).
- *Flattening*: Removing nested conjunctions where possible.

G ALFWORLD DETAILS

G.1 ALFWORLD TASKS

ALFWorld simulates a typical household environment and focuses on daily embodied AI tasks. Table 4 lists the supported task types along with their corresponding goal templates.

Table 4: Task-types and templated goal descriptions in ALFWorld.

Task-type	Templates
Pick & Place	(a) put a {obj} in {recep}.
	(b) put some {obj} on {recep}.
Examine in Light	(a) look at {obj} under the {lamp}.
	(b) examine the {obj} with the {lamp}.
Clean & Place	(a) put a clean {obj} in {recep}.
	(b) clean some {obj} and put it in {recep}.
Heat & Place	(a) put a hot {obj} in {recep}.
	(b) heat some {obj} and put it in {recep}.
Cool & Place	(a) put a cool {obj} in {recep}.
	(b) cool some {obj} and put it in {recep}.
Pick Two & Place	(a) put two {obj} in {recep}.
	(b) find two {obj} and put them {recep}.

- **Pick & Place** (e.g., “put a plate on the coffee table”) — the agent must find an object of the desired type, pick it up, find the correct location to place it, and put it down there.
- **Examine in Light** (e.g., “examine a book under the lamp”) — the agent must find an object of the desired type, locate and turn on a light source with the desired object in-hand.
- **Clean & Place** (e.g., “clean the knife and put in the drawer”) — the agent must find an object of the desired type, pick it up, go to a sink or a basin, wash the object by turning on the faucet, then find the correct location to place it, and put it down there.
- **Heat & Place** (e.g., “heat a mug and put on the coffee table”) — the agent must find an object of the desired type, pick it up, go to a microwave, heat the object by turning on the microwave, then find the correct location to place it, and put it down there.
- **Cool & Place** (e.g., “put a cool bottle on the countertop”) — the agent must find an object of the desired type, pick it up, go to a fridge, put the object inside the fridge and cool it, then find the correct location to place it, and put it down there.
- **Pick Two & Place** (e.g., “put two pencils in the drawer”) — the agent must find an object of the desired type, pick it up, find the correct location to place it, put it down there, then look for another object of the desired type, pick it up, return to the previous location, and put it down there with the other object.

G.2 Example Trajectory

```
You are in the middle of a room. Looking quickly around you, you see a
drawer 15, a drawer 19,
a drawer 7, an armchair 1, ... and a drawer 10.
Your task is to: find two remotecontrol and put them in armchair.
> go to sidetable 2
You arrive at loc 34. On the sidetable 2, you see a remotecontrol 1.
> take remotecontrol 1 from sidetable 2
You pick up the remotecontrol 1 from the sidetable 2.
> go to armchair 1
You arrive at loc 1. On the armchair 1, you see nothing.
> put remotecontrol 1 in/on armchair 1
You put the remotecontrol 1 in/on the armchair 1.
> go to sofa 1
You arrive at loc 2. On the sofa 1, you see a newspaper 1, a pillow 1, and
a remotecontrol 2.
> take remotecontrol 2 from sofa 1
You pick up the remotecontrol 2 from the sofa 1.
> go to armchair 1
You arrive at loc 1. On the armchair 1, you see a remotecontrol 1.
> put remotecontrol 2 in/on armchair 1
You won!
```

H Entailment Details

H.1 Entailment Design

In this section we outline how we misaligned the action model and the goal for both AFLWorld and Blocksworld domain.

ALFWorld Entailment Predicates Instead of using symbolic predicates such as, we define each action effect and precondition directly in NL and allow multiple entailed variants. Hard string matching is replaced with entailment sets as follows:

$$\begin{aligned}
\text{handEmpty} &: \{\text{“the agent's hand is empty”}, \text{“the agent is not holding anything”}\} \\
\text{in}(o, r) &: \{\text{“}o\text{ is in }r\text{”}, \text{“}o\text{ is inside }r\text{”}, \text{“}o\text{ is stored in }r\text{”}\} \\
\text{hot}(o) &: \{\text{“}o\text{ is heated”}, \text{“}o\text{ is hot”}, \text{“}o\text{ is baked”}\} \\
\text{washed}(o) &: \{\text{“}o\text{ is washed”}, \text{“}o\text{ is clean”}, \text{“}o\text{ is cleaned”}\} \\
\text{cooled}(o) &: \{\text{“}o\text{ is chilled”}, \text{“}o\text{ is cool”}, \text{“}o\text{ is cooled”}\} \\
\text{holding}(o) &: \{\text{“the agent is holding }o\text{”}, \text{“the agent possesses }o\text{”}, \text{“the agent has }o\text{ in possessing”}\} \\
\text{on}(r) &: \{\text{“}r\text{ is turned on”}, \text{“}r\text{ is on”}, \text{“}r\text{ is switched on”}\}
\end{aligned}$$

Blocksworld Entailment Predicates). We express each predicate directly in NL and allow multiple entailed variants rather than relying on canonical symbols. The entailment sets are:

$$\begin{aligned}
\text{handEmpty} &: \{\text{“the agent is not holding any objects”}, \text{“the agent is not holding anything”}\} \\
\text{clear}(b) &: \{\text{“block }b\text{ has nothing on top of it”}, \text{“no block is on }b\text{”}\} \\
\text{onTable}(b) &: \{\text{“block }b\text{ sits directly on the table”}, \text{“}b\text{ is on the table”}\} \\
\text{on}(b_1, b_2) &: \{\text{“block }b_1\text{ is directly above block }b_2\text{”}, \text{“}b_1\text{ on }b_2\text{”}\} \\
\text{holding}(b) &: \{\text{“the agent possesses block }b\text{”}, \text{“the agent is holding }b\text{”}\}
\end{aligned}$$

H.2 Natural Language Description of Action Models

We convert each action schema into a natural language description that specifies its preconditions and effects. These NL descriptions are provided to all LLM- and VLM-based models. To test alignment, we intentionally modify the descriptions to use phrases that entail the goal but do not exactly match the goal string, thereby requiring explicit entailment reasoning. Below we provide examples of the NL descriptions of both the ALFWorld and Blocksworld action models.

Natural Language ALFWorld Action Model :

```
The pickup action requires the agent's hand to be empty and results in the
agent holding the target object. The put action requires the agent to
already be holding an object and allows the object to be placed inside or
on top of another receptacle, after which the agent's hand becomes empty.
The heat action requires the agent to be holding an object and to be at a
heating device, producing the effect that the object becomes hot or baked.
The wash action requires the agent to be holding the object and to be at a
washing device, resulting in the object becoming washed. The chill action
requires the agent to be holding the object and to be at a chilling device,
resulting in the object becoming chilled. The light action requires the
agent to be holding the object and to be at a lighting device, producing
the effect that the object becomes illuminated. Finally, device state can
be toggled by turn on and turn off, which require the agent to be at the
device and result in it being switched on or off
```

Natural Language Blocksworld Action Model :

```
The pickup action requires the agent's hand to be empty and results in the
agent holding the target object. The put action requires the agent to
already be holding an object and allows the object to be placed inside or
on top of another receptacle, after which the agent's hand becomes empty.
The heat action requires the agent to be holding an object and to be at a
heating device, producing the effect that the object becomes hot or baked.
The wash action requires the agent to be holding the object and to be at a
washing device, resulting in the object becoming washed. The chill action
requires the agent to be holding the object and to be at a chilling device,
resulting in the object becoming chilled. The light action requires the
agent to be holding the object and to be at a lighting device, producing
the effect that the object becomes illuminated. Finally, device state can
be toggled by turn on and turn off, which require the agent to be at the
device and result in it being switched on or off
```

I Design Details of NL-PDDL ALFWorld and Blocksworld Implementation

I.1 ALFWorld

In addition to the core NL-PDDL planner, our ALFWorld agent consists of an VLM-based observation parser, a knowledge base (KB), and an LLM-based object grounder. The observation parser extracts a list of object names from either text or image input. We use Gemini-2.0-Flash (Team et al., 2023) for image observations parsing. After obtaining the object names, we use GPT-4o to instantiate these into NL-predicates and update the agent's KB. The KB is implemented using PyDatalog (Carbonnelle, 2024). Whenever new knowledge is acquired, the agent queries subgoals generated by the NL-PDDL regression planner to determine a feasible plan to execute.

Observation Parser The observation parser is responsible for extracting candidate object names and their types from either textual descriptions or visual observations. In the ALFWorld Text setting, object names are directly provided in the environment description, so we simply parse this list to extract object references. In the ALFWorld Vision setting, the parser receives RGB frames from a simulated first-person camera. Here, we use Gemini-2.0-Flash (Team et al., 2023), a VLM capable of open-vocabulary object detection, to generate bounding boxes around objects of interest. Gemini is prompted with relevant object descriptions (e.g., "Find a cup", "Find something that can wash the cup") derived from subgoals generated by the NL-PDDL planner, and it returns a list of detected objects with bounding boxes, labels, and confidence scores. The resulting object list serves as the input to the grounding stage but does not by itself produce predicates or symbolic facts.

Object Grounder The object grounder bridges the gap between raw object names and the structured predicates required by the NL-PDDL planner. Given the list of parsed object tokens and their textual or visual context, we use GPT-4o to generate grounded natural language predicates that align with our lifted planning formalism. For example, given the input "mug is cleaned," the grounder infers the predicate `clean(mug_1)`. The grounder also infers action-relevant relations such as `canClean(sink_1, apple_1)` when provided with relevant context. This ensures that grounded facts are consistent with both the agent's environment and the symbolic schema expected by the planner.

Knowledge Base The knowledge base (KB) stores grounded predicates and supports logical reasoning over them. We implement the KB using PyDatalog (Carbonnelle, 2024), a lightweight logic programming library for Python that supports declarative predicate logic, variable binding, and rule-based inference. During plan execution, the planner queries the KB to check whether a lifted subgoal can be satisfied with the current facts. If the KB entails a subgoal, we bind the variables in both the subgoal and the associated actions, and then execute the corresponding plan. This allows the planner to maintain consistency between abstract subgoals and grounded execution.

I.2 BLOCKSWORLD

For the Blocksworld domain, we convert the original PDDL goal into a corresponding NL-PDDL formulation. We follow the methodology of Valmeekam et al. (2023a) to ensure consistency between symbolic and natural language representations. The lifted regression planner then produces a sequence of subgoals with a fixed horizon of 10, which are grounded and validated sequentially. Each subgoal and the agent's initial state are translated into PyDatalog and checked for logical consistency with the original PDDL specification. Once all subgoals are validated, we use the standard PDDL validator from Muise et al. (2022) to ensure correctness of the complete plan.

I.3 LLM PROMPT FOR TYPE CONSISTENCY CHECK

```
ROLE:
   You are a helper agent in a common household setting.
   You are checking TYPE ENTAILMENT between two predicates' term types.
INSTRUCTION:
   1. If the candidate's term type set implies or matches the target's term
   type set for each corresponding term, answer YES.
   2. Consider synonyms and common-sense subtype relations (e.g.,
   'vegetable' entails 'food').
   3. If information is unknown, be conservative and answer NO unless it's
   very likely.
INPUT:
   - Target term types: {$predicates_in_action_model_type$}
   - Candidate term types: {$misaligned_goal_type$}
OUTPUT FORMAT:
   - Line 1: exactly YES or NO.
   - Line 2: Reason.
RESPONSE:
```

```
ROLE:
   You are a helper agent in a common household setting.
QUESTION:
   - if you know Predicate 2 "{$action_model_predicate$}" is true, can you
   imply Predicate 1 "{$misaligned_goal_predicate$}" is true?.
   - Respond with exactly "YES" if you think the statement is generally
   implied
   - Respond with "NO" if you think the statement is generally false
INPUT:
   - Predicate 1: "{$misaligned_goal_predicate$}"
   - Predicate 2: "{$action_model_predicat$e$}"
INSTRUCTION:
   1. Use the definition of the predicates to determine if Predicate 2
   implies Predicate 1.
   2. You know the following background to determine the specific
   information of the objects within Predicate 1 and Predicate 2: $goal
   predicates$
   3. When determining the response, consider the meaning of the Predicate
   1 and Predicate 2 with the type of the specific object each refers to in
   common contexts.
   4. Be creative and think outside the box. If there is just a typo
   between the two predicates, you should say Yes.
OUTPUT FORMAT:
   - Line 1: exactly YES or NO.
   - Line 2: Reason.
RESPONSE:
```

J WORKS ON TRANSLATING NL TO PDDL

PDDL has long been established as the primary standard for defining planning domains and problems in AI Gerevini (2020); Haslum et al. (2019). However, authoring accurate PDDL domains and problem specifications is a resource-intensive process that requires human expertise. To facilitate this process, a growing body of work has explored translating NL descriptions of planning domains and problems into PDDL.

Early efforts include NLtoPDDL Miglani & Yorke-Smith (2020), a pipeline that leverages readily available NL data by combining pre-trained contextual embeddings with Deep Reinforcement Learning (DRL) techniques previously used to extract structured plans from NL. Another example is FPTCP Huo et al. (2020), which constructs a ternary template of NL sentences to extract actions and their associated objects in a human-in-the-loop framework.

With the emergence of large language models (LLMs), their strong language understanding and commonsense reasoning abilities have been used to further facilitate PDDL construction from NL. For instance, Xie et al. (2023) uses the in-context learning capabilities of LLMs to translate NL domains and problems into PDDL via few-shot examples, demonstrating that using LLMs solely for translation and delegating the planning step to a PDDL solver yields superior results compared to direct LLM-based planning. Similarly, Smirnov et al. (2024) employs an LLM to generate PDDL plans, but also introduces LLM-based consistency checks and error-correction loops to improve plan quality. More recently, advanced approaches such as Ada **?** have been proposed. Rather than translating a pre-existing instruction or domain manual into a single PDDL file, Ada interactively learns adaptive planning representations. In this framework, high-level action abstractions and low-level controllers are jointly adapted to a given domain of planning tasks, guided by NL inputs.

Although these seminal works significantly improve the usability of PDDL, they are all limited to translating NL inputs into PDDL and then relying on an existing PDDL planner—a process that can be error-prone due to the inherent limitations of LLMs and the auxiliary techniques involved. In contrast, NL-PDDL directly extends the PDDL framework to natively support NL specifications, thereby eliminating the need for NL-to-PDDL translation. Moreover, prior approaches primarily focus on generating and refining closed-world domains, whereas NL-PDDL supports open-world planning with incomplete domain knowledge.

K NUMBER OF UNDEFINED PREDICATES IN ALFWORLD

In practical scenarios, misalignment between an agent's action model and the underlying task goals is common. Consequently, it is essential for a symbolic planner to incorporate LLM-based reasoning when necessary—a capability that NL-PDDL is explicitly designed to provide.

As shown in Table 5, In the ALFWorld domain, a total of 583 unique predicates appear across all tasks, with an average of 52.26 predicates per task. Yet the action model defines only 22 predicates—approximately 3.7% of all predicates observed in our experiments. The remaining predicates arise dynamically from the user's instructions and the agent's interaction with the environment.

Because these predicates are not predefined, existing symbolic planning approaches that require a fixed, closed-world predicate set become infeasible. In contrast, NL-PDDL is the only formal framework that removes the need for a predefined predicate vocabulary by leveraging LLM-based entailment reasoning to interpret and ground predicates on demand.

Table 5: Predicate statistics for the ALFWorld domain.

Statistic	Value
Total Number of Predicates	583
Predefined Predicates	22
Predicates per Task	52.26

L ENTILAMENT CHECK

We conducted an additional ablation study (cf. Table 6) to evaluate the correctness of the LLM when performing predicate-level affordance reasoning over natural-language predicates. We randomly selected 20 predicates from ALFWorld and asked a human annotator to manually provide three sentences that should be entailed by each predicate and three sentences that should not be entailed. This yielded 120 entailment–non-entailment pairs in total. We then evaluated all pairs using the NL-PDDL entailment procedure to assess the model's precision and robustness in predicate-level reasoning.

Table 6: Classification statistics for the ALFWorld predicate evaluation.

Metric	Number of Instances
True Positive	60
True Negative	58
False Positive	0
False Negative	2
Total Accuracy	98.3

M LLM CALLS AND LATENCY

We randomly select 20 goals from *Misaligned BlockWorld* and *Misaligned ALFWorld* for each depth and report the mean and 95% confidence interval of the total number of LLM calls in Table 7. These totals include both local cache hits (i.e., we store all previously encountered LLM entailments in a global cache shared across goals) and actual API calls.

Table 7: LLM call and entailment statistics across domains.

Domain	ALFWorld with Misalignment	Misalignment BlocksWorld
LLM API Call	50.4 ± 12.9	73.2 ± 23.4
Total Entailments Reasoning	221.5 ± 155.6	1263 ± 435.7
Avg Latency per Call	1.8648 sec	1.2443 sec