Observe, Plan, and Train – Towards a Fully Automated Approach of Harnessing Large Language Models in Language-Informed Reinforcement Learning

Anonymous submission

Abstract

In language-informed reinforcement learning, the prevalent approach, known as Language Reward Shaping (LRS), converts language data into scalar reward signals for training policy networks. However, conventional LRS technique exhibits notable shortcomings that can diminish learning efficiency. Addressing these issues, we introduce a novel and fully automated framework. It extends LRS by leveraging large language models (LLMs) to translate task domain and problem descriptions into the Planning Domain Definitions Language (PDDL). The PDDL configuration is used to generate a high-level action plan. Each specific high-level action is executed through modular control policies trained by LRS model. This approach has proven effective in the challenging environment of the Atari game, Montezuma's Revenge, demonstrating superior performance over the traditional LRS approach.

1 Introduction

Reinforcement Learning (RL) experiences a significant improvement in learning efficiency through the incorporation of language information into learning algorithms (Luketina et al. 2019). An agent capable of extracting and utilizing knowledge from language information gains auxiliary guidance for decision-making, preventing it from having a whole avenue of exploration. To harness this advantage, the prevalent approach known as Language Reward Shaping (LRS) is employed. This approach involves training a reward function that provides rewards when agents' behaviors closely align with the provided language instructions. (Kaplan, Sauer, and Sosa 2017; Goyal, Niekum, and Mooney 2019; Wang et al. 2019; Du et al. 2023).

Notwithstanding its extensive adoption, LRS faces certain limitations that hinder its effectiveness. Firstly, the scalar reward representation struggles to preserve the chronological order in longer textual content (Pham et al. 2020). Secondly, converting high-dimensional language data into scalars inevitably results in the omission of subtler nuances and introduces ambiguity (Gedeon and Harris 1992; Huang, Lipovetzky, and Cohn 2023). Lastly, the technique solely makes use of direct task instructions while falling short in utilizing more general textual information, such as manuals.

To overcome these challenges, we propose a novel and fully automated approach to better harness language data in RL paradigm. Our approach employs large language models (LLMs) to transform domain descriptions and task-specific instructions into the Planning Domain Definitions Language (PDDL). Following this transformation, a classical planner interprets the PDDL and yields a coherent high-level plan. Separate modular control policies are then constructed for each high-level action in the plan. These modular policies still incorporate LRS method, with auxiliary rewards being obtained from a pretrained LRS model.

This research contributes to the field by providing a more efficient and effective approach to language-informed reinforcement learning. The core objective of this paper is to outline our methodology, present the implementation results, with a specific focus on Montezuma's Revenge, a challenging and well-established benchmark environment that provides publicly available textual information.

2 Related Work

LLMs in Sequential Decision-Making: with the advancement of LLMs' reasoning capabilities, their role in sequential decision-making has been a dominating research topic (Li et al. 2023). Previous work in this area leveraged LLMs to generate sound plans in natural language format, subsequently grounding these plans into defined actions in the world model (Xiang et al. 2023; Huang et al. 2023).

LLMs and PDDL Integration: an intriguing yet relatively underexplored area of research lies in the integration of LLMs and classical AI planners. The studies conducted by Silver et al. (2022) and Xie et al. (2023) examined the application of LLMs in planning tasks based on descriptions written in either natural language or PDDL format. Nevertheless, Valmeekam et al. (2022) raised criticism regarding the plausible but inconsistent reasoning of LLMs. In response, instead of directly relying on the reasoning capabilities of LLMs, Guan et al. (2023) and Liu et al. (2023) proposed translating natural language descriptions into structured format, which are subsequently processed by an external AI planner to generate action plans. However, these studies primarily focus on high-level planning, leaving room for exploration in grounding plans into the low-level control interactions with the environment.

Language Reward Shaping (LRS): it has been a well-accepted approach in the field of language-informed reinforcement learning. Instead of taking text embeddings as policy model inputs, LRS trains a reward function that gives

auxiliary rewards when agent behaviours align with the text instructions. This is particularly useful in environments with sparse reward signals. However, conventional LRS suffers from issues including the inability to maintain chronological order, information loss caused by compression, and limited utility of language data. Previous successful applications of LRS were constrained to environments with smaller discrete action spaces (Du et al. 2023), short atomic sentence instructions (Goyal, Niekum, and Mooney 2019), or required intricate domain-specific pretraining (Kaplan, Sauer, and Sosa 2017).

In this study, we seek to enhance LRS technique and present "Observe, Plan and Train" (OPT) framework in the field of language-informed reinforcement learning, covering from high-level planning to low-level control policy learning for sequential decision-making tasks.

3 Task Overview

In this study, we consider tasks in the form of Goal Markov Decision Process (Goal MDP) (Ramırez and Geffner 2011), which can be written as a tuple given by

- A non-empty state space, denoted by S.
- A non-empty goal state subset, represented as $S_G \subseteq S$.
- An initial state s_0 belonging to S.
- A set of possible actions, A.
- A transition function, $T : S \times A \to S$, delineating the system dynamics.
- A reward function, $R_{\text{env}} : S \times A \to \mathbb{R}$.

The objective of an agent is to train a policy π, that performs a sequence of actions $\{a_0, ..., a_t\}, a \in A$ – leading to the goal states $g \in S_G$ while maximizing the cumulative rewards $\sum^t \gamma^t R_{\text{env}}(s_t, a_t)$. LRS introduces a reward shaping function Φ, which rewards agents when their policy aligns with the expert policy π^* described in natural language. To include the natural language information, we define a natural language projector $\mathcal{Z}$ that maps elements to their corresponding natural language descriptions. Under LRS, we obtain a non-Markovian reward function $R = R_{\text{env}} + \Phi$, where $\Phi = sim(\pi, \hat{\pi^*})$. Here, $\hat{\pi^*} \sim (\cdot|\mathcal{Z}(\pi^*))$ denotes the inferred expert policy grounded in its natural language description, and $sim(\pi, \hat{\pi^*})$ quantifies how closely the agent's current policy mirrors the target expert policy.

The Planning Domain Definition Language (PDDL): the presence of auxiliary language information not only facilitates LRS techniques but also paves the way for integrating classical AI planning with traditional RL methods. The planning model aligns with the same Goal MDP framework, though it typically assumes a deterministic transition function. PDDL (Aeronautiques et al. 1998) offers a standard and compact way for task representation. A task in PDDL is expressed as a tuple $\langle \mathcal{D}, \mathcal{I} \rangle$:

1. **Domain ($\mathcal{D}$)**: The domain, represented as $\mathcal{D} = \langle \mathcal{F}, \mathcal{B} \rangle$, encapsulates:
   (a) **Predicates ($\mathcal{F}$)**: These represent properties or relations in the planning domain that can either hold true or false.
   (b) **Action Schema Set ($\mathcal{B}$)**: Each action schema $\beta \in \mathcal{B}$ is a construct $\langle par, pre, \mathit{eff} \rangle$, where:
      i. par details the parameters or arguments of the action.
      ii. pre denotes the preconditions—sets of predicates that delineate the states under which the action can be applied.
      iii. eff designates the effects, being conjunctive logical expressions, indicating the transformation of the state when the action is executed. Notably, literals in both the preconditions and effects can exclusively employ parameter symbols declared within the action's parameters.
2. **Instance ($\mathcal{I}$)**: Specified as $\mathcal{I} = \langle \Delta, I, G \rangle$, the instance comprises:
   (a) **Objects (Δ)**: A collection of entities used to instantiate parameters in the domain. $\mathcal{F}$ and $\mathcal{B}$ can be instantiated with Δ resulting in a set of ground instances of predicates (i.e., facts) F, high-level actions B and state space $S = 2^F$.
   (b) **Initial State (I)**: It catalogues the facts that are true at the task's initial state s_0.
   (c) **Goal Conditions (G)**: These provide the conjunction of facts required to discern when a goal state S_G is achieved.

4 Method

The OPT framework adopts a two-phase approach, consisting of a *High-Level Planning Phase* and a *Low-Level Control Learning Phase*. We explain the steps involved in each phase in below.

4.1 High-Level Planning Phase

Existing work in language-informed RL has generally limited to detailed instructions, often failing to leverage general descriptive text (Sumers et al. 2022). The key challenge lies in the difficulty of constructing a straightforward projection from high-level information onto agent policy. Our OPT framework aims to bridge this gap by to employing Large Language Models (LLMs) to convert descriptive language into high-level planning configurations in PDDL format.

PDDL Generation from Descriptive Text Although recent work has criticised LLMs for their ability of reasoning from first principles (Kambhampati et al. 2023), they have demonstrated the capability in solving the *sequence to sequence transformation problem* (Mirchandani et al. 2023). Therefore, based on the hypothesis that LLMs can be a good translator, we take domain descriptions $Z(\mathcal{D})$ (e.g., manuals) and instance descriptions $\mathcal{Z}(\mathcal{I})$ (e.g., instructions) as the prompt content. A pretrained LLM is then used to transform natural language descriptions into structured PDDL format $\langle \mathcal{D}, \mathcal{I} \rangle$. More details are in Appendix A. We also implement Chain-of-Thought (CoT) prompting style (Wei et al. 2022; Wang et al. 2023) to ensure consistency and correctness in LLMs' output. See examples in Appendix B.

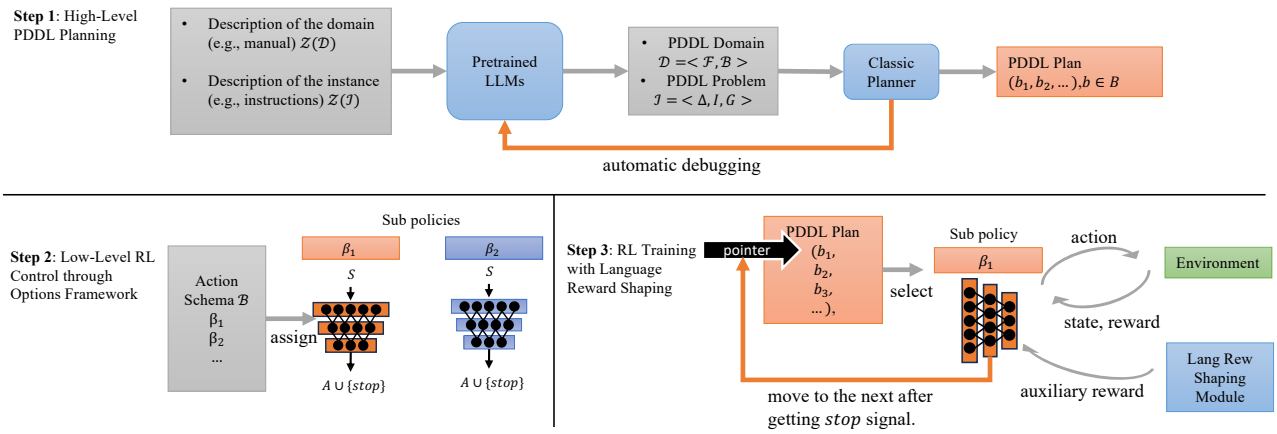

Figure 1: OPT framework overview: (Step 1) A pretrained LLM translate task descriptions into the structured PDDL format. Following this, a classic planner derive a high-level action plan from the PDDL configuration. (Step 2) Separate sub policy modules, termed *options module*, are established for each action schema β within the action schema set $\mathcal{B}$. (Step 3) Starting from the first high-level action b_1, each of the associated sub policy module π_{β_i} undergoes reinforcement learning with auxiliary reward from a pretrained LRS model. It will proceed to the next high-level action b_{i+1} and its associated sub policy module once the preceding module emits a $stop$ signal. See text for more details.

Automated Interactive Debugging LLMs may produce PDDL configurations with syntax errors. In response, we implement an automatic debugging loop that iteratively rectifies the PDDL output by feeding the error message from the classic planner back to LLMs. The loop is repeated until the planner successfully acquires a valid high-level action plan $\Omega = (b_1, b_2, ...), b \in B$.

4.2 Low-Level Control Learning Phase

During the RL of the low-level control policies, LLMs are not more activated due to their heavy computations, which can introduce latency issues in RL. After getting the high-level action plan, we use *options* framework (Sutton, Precup, and Singh 1999) to ground the high-level actions into low-level control.

***Options* Framework for Low-Level Control** Every action schema β is associated with a subpolicy model π_β. The range of the subpolicy is an augmented action space that combines the low-level action set with a special $stop$ action, denoted as $A \cup \{stop\}$ (see step 2 in Figure 1). A task-specific policy Π_Ω is formed by concatenating a sequence of associated subpolicies that correspond to the predefined high-level action plan $\Omega = (b_1, b_2, ...)$, where each action b is an instantiated version of the associated action schema. Π_Ω maintains a index pointer i, initialised to 1, and calls upon the subpolicy π_{β_i} to generate low-level actions, where β_i refers the corresponding action schema of action instance b_i. Once the $stop$ signal is emitted, the index pointer i is incremented, shifting control to the next subpolicy $\pi_{\beta_{i+1}}$.

The Π_Ω can be viewed as a Markov chain that operates over the state space $S \times B$ with the following transitions:

$$(s, a_i) \rightarrow \begin{cases} (s', b_i) & \text{with pr.} \quad \sum_{a \in A} \pi_{\beta_i}(a|s) \cdot P(s'|s, a) \\ (s, b_{i+1}) & \text{with pr.} \quad \pi_{\beta_i}(stop|s) \end{cases} \tag{1}$$

The Π_Ω is designed to preserve the chronological order of agent behaviours at high-level. This feature enhances the stability of the system, especially when faced with lengthier sequential decision-making steps. This constitutes a marked improvement over conventional language-informed RL models, which often struggle when instruction sets get lengthier. Moreover, by leveraging high-level task decomposition through classical planning, our approach reduces the demand on RL-based control policies to handle extensive long-horizon reasoning. This shift allows these policies to concentrate primarily on intricate, low-level control aspects.

Auxiliary Language Reward for Subpolicy Learning One remaining challenge is to train modular subpolicies to emit $stop$ signals and shift control at a right juncture. Traditional methods like supervised behaviour cloning can be costly for annotating trajectories. For that reason, our OPT framework proposes to re-use LRS technique to offer an indirect supervisory signal for training subpolicies on when to stop. Specifically, the LRS function Φ employs a visual captioner, labeled as C, to describe trajectories τ generated by the subpolicy. It outputs the semantic similarity score of the natural language description of the associated high-level action step and the caption of the trajectory. Mathematically, it can be expressed as:

$$\Phi(b, \tau) = sim(Z(b), C(\tau)) \tag{2}$$

This approach circumvents the issue of vanilla LRS method, which tended to reward partially matched trajectories when the instruction content gets lengthier. By segmenting a big task into smaller, more manageable subgoal unit, our OPT framework guarantees that each auxiliary reward calculation only involves laconic textual input. This reduce the risk of false positive rewards since the compression rate of transforming textual data into scalar similarity score is reduced.

Algorithm 1: Modular Subpolicies RL with LRS

Require: : $S, S_G, s_0, A, T, R_{env}, \Phi, \mathcal{B}, \Omega = (b_1, b_2, ...b_n)$

▷ *Initialisation* ◁
$N \leftarrow$ Number of Episodes
Initialise global replay buffer B_G
$\Pi_\Omega \leftarrow [\,]$ ▷ *init Π_Ω as empty list*
for all $\beta \in \mathcal{B}$ **do**
    Randomly initialize policy network π_β for each action schema β
for all $b \in \Omega$ **do**
    $\Pi_\Omega \leftarrow \Pi_\Omega + [\pi_{\beta_b}]$ ▷ *append associated subpolicies*
▷ *Training* ◁
for $n = 1, \ldots, N$ **do**
    $i \leftarrow 1$ ▷ *reset index pointer*
    while $i \leq n$ and S_G not reached **do**
        $\pi \leftarrow \Pi_\Omega[i]$ ▷ *select policy at index i*
        Initialise local replay buffer B_L
        repeat
            Select and execute action from the current policy $a \sim \pi_{\beta_i}(\cdot|s)$
            Store transition (s, a, s', r) in both B_G and B_L
        until *stop* action or S_G is reached
        ▷ *modify B_L by adding auxiliary language reward* ◁
        for all $(s, a, s', r) \in B_L$ **do**
            $r \leftarrow r + \Phi(b_i, B_L)$
        ▷ *Train locally* ◁
        Sample minibatch transitions from B_L
        Update the local policy π using RL algorithms such as Actor Critic
        $i \leftarrow i + 1$
    ▷ *Train globally* ◁
    Sample minibach transitions from B_G
    for all $\pi \in \Pi_\Omega$ **do**
        Update policy π using RL algorithms such as Actor Critic
return Π_Ω

Algorithm 1 shows pseudocode for subpolicy RL with LRS.

References

Aeronautiques, C.; Howe, A.; Knoblock, C.; McDermott, I. D.; Ram, A.; Veloso, M.; Weld, D.; SRI, D. W.; Barrett, A.; Christianson, D.; et al. 1998. Pddl— the planning domain definition language. *Technical Report, Tech. Rep.*

Du, Y.; Watkins, O.; Wang, Z.; Colas, C.; Darrell, T.; Abbeel, P.; Gupta, A.; and Andreas, J. 2023. Guiding Pretraining in Reinforcement Learning with Large Language Models. arXiv:2302.06692.

Gedeon, T.; and Harris, D. 1992. Progressive image compression. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 4, 403–407. IEEE.

Goyal, P.; Niekum, S.; and Mooney, R. J. 2019. Using natural language for reward shaping in reinforcement learning. *arXiv preprint arXiv:1903.02020*.

Guan, L.; Valmeekam, K.; Sreedharan, S.; and Kambhampati, S. 2023. Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning. *arXiv preprint arXiv:2305.14909*.

Huang, S.; Lipovetzky, N.; and Cohn, T. 2023. A Reminder of its Brittleness: Language Reward Shaping May Hinder Learning for Instruction Following Agents. *arXiv preprint arXiv:2305.16621*.

Huang, W.; Xia, F.; Shah, D.; Driess, D.; Zeng, A.; Lu, Y.; Florence, P.; Mordatch, I.; Levine, S.; Hausman, K.; et al. 2023. Grounded decoding: Guiding text generation with grounded models for robot control. *arXiv preprint arXiv:2303.00855*.

Kambhampati, S.; Valmeekam, K.; Marquez, M.; and Guan, L. 2023. On the Role of Large Language Models in Planning. Tutorial presented at the International Conference on Automated Planning and Scheduling (ICAPS), Prague.

Kaplan, R.; Sauer, C.; and Sosa, A. 2017. Beating atari with natural language guided reinforcement learning. *arXiv preprint arXiv:1704.05539*.

Li, W.; Luo, H.; Lin, Z.; Zhang, C.; Lu, Z.; and Ye, D. 2023. A survey on transformers in reinforcement learning. *arXiv preprint arXiv:2301.03044*.

Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; Biswas, J.; and Stone, P. 2023. LLM + P: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Luketina, J.; Nardelli, N.; Farquhar, G.; Foerster, J.; Andreas, J.; Grefenstette, E.; Whiteson, S.; and Rocktäschel, T. 2019. A survey of reinforcement learning informed by natural language. *arXiv preprint arXiv:1906.03926*.

Mirchandani, S.; Xia, F.; Florence, P.; Ichter, B.; Driess, D.; Arenas, M. G.; Rao, K.; Sadigh, D.; and Zeng, A. 2023. Large Language Models as General Pattern Machines. *arXiv preprint arXiv:2307.04721*.

Pham, T. M.; Bui, T.; Mai, L.; and Nguyen, A. 2020. Out of Order: How important is the sequential order of words in a sentence in Natural Language Understanding tasks? *arXiv preprint arXiv:2012.15180*.

Ramırez, M.; and Geffner, H. 2011. Goal recognition over POMDPs: Inferring the intention of a POMDP agent. In *IJCAI*, 2009–2014. IJCAI/AAAI.

Silver, T.; Hariprasad, V.; Shuttleworth, R. S.; Kumar, N.; Lozano-Pérez, T.; and Kaelbling, L. P. 2022. PDDL planning with pretrained large language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Sumers, T.; Hawkins, R.; Ho, M. K.; Griffiths, T.; and Hadfield-Menell, D. 2022. How to talk so AI will learn: Instructions, descriptions, and autonomy. *Advances in Neural Information Processing Systems*, 35: 34762–34775.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211.

Valmeekam, K.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2022. Large Language Models Still Can't Plan (A Benchmark for LLMs on Planning and Reasoning about Change). *arXiv preprint arXiv:2206.10498*.

Wang, J.; Sun, Q.; Chen, N.; Li, X.; and Gao, M. 2023. Boosting Language Models Reasoning with Chain-of-Knowledge Prompting. *arXiv preprint arXiv:2306.06427*.

Wang, X.; Huang, Q.; Celikyilmaz, A.; Gao, J.; Shen, D.; Wang, Y.-F.; Wang, W. Y.; and Zhang, L. 2019. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 6629–6638.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837.

Xiang, J.; Tao, T.; Gu, Y.; Shu, T.; Wang, Z.; Yang, Z.; and Hu, Z. 2023. Language Models Meet World Models: Embodied Experiences Enhance Language Models. *arXiv preprint arXiv:2305.10626*.

Xie, Y.; Yu, C.; Zhu, T.; Bai, J.; Gong, Z.; and Soh, H. 2023. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*.

A Few-shot Prompting Example For Converting Descriptive Language into PDDL

Prompt: You are defining task domain represented in PDDL format. We will do this step by step, first we will ask you to summarize the domain using bullet points in natural language, after that, we will ask you to convert each bullet points into PDDL format.
Here is one example from the "miconic" domain. It's a simple model of an elevator system. Here is a summary of the domain:

- Objects: There are two types of objects in this domain: passengers and floors.
- Predicates: The domain has several predicates to describe the state of the world:
  - 'origin': This predicate specifies the starting floor of a given passenger.
  - 'destin': This predicate indicates the destination floor of a given passenger.
  - 'above': This indicates that a particular floor is above another floor.
  - ...
- Actions: The domain has four actions that can change the state of the world:
  - 'board': A passenger boards the elevator. The precondition for this action is that the elevator is at the same floor as the origin floor of the passenger. The effect is that the passenger is now considered to have boarded.
  - 'depart': A passenger departs from the elevator. The precondition is that the elevator is at the passenger's destination floor and the passenger has boarded the elevator. The effect is that the passenger is no longer considered to be on board and is now considered to be served.
  - ...

The corresponding domain PDDL format is

```
(define (domain miconic)
  (:requirements :strips)
  (:types passenger - object
          floor - object
  )

(:predicates
(origin ?person - passenger ?floor - floor)
(destin ?person - passenger ?floor - floor)
(above ?floor1 - floor  ?floor2 - floor)
(boarded ?person - passenger)
(served ?person - passenger)
(lift-at ?floor - floor)
)

(:action board
  :parameters (?f - floor ?p - passenger)
  :precondition (and (lift-at ?f)
                     (origin ?p ?f)
                )
  :effect (and (boarded ?p)
          )
)

...
)
```

Now, I will share with you the manual of *<YOUR-TASK-DOMAIN>*:
– Start of the manual –
<Content of YOUR-TASK-DOMAIN> ...
– End of the manual –
Please response 'yes' if you are ready to summarize *<YOUR-TASK-DOMAIN>* domain.

B Chain of Thought Example

User: Please response yes if you are ready to summarize *<YOUR-TASK-DOMAIN>* domain.
LLM: Yes, I am ready to summarize the *<YOUR-TASK-DOMAIN>* domain. Here is my summary:

- Objects: *<natural language descriptions of the objects>*.
- Predicates: *<natural language descriptions of the predicates>*.
- Actions: *<natural language descriptions of the actions>*.

User: Please generate *:types* section in PDDL format according to the summary.

LLM: Sure, based on the summary the *:types* section in PDDL format would be: *<PDDL format of the types>*.

User: Please generate *:predicates* section in PDDL format according to the summary.

LLM: Sure, based on the summary the *:predicates* section in PDDL format would be: *<PDDL format of the predicates>*.

User: Please generate *:action* section in PDDL format according to the summary.

LLM: Based on the summary, here are examples of *:action* sections in PDDL format: *<PDDL format of the action>*.

User: Concatenate the generated PDDL together to form a complete domain PDDL file.

LLM: Sure, here is the complete domain PDDL file based on the provided information: *<PDDL Domain File of YOUR-TASK-DOMAIN>*.

User: The instruction of *<ONE-OF-YOUR-TASK-INSTANCE>* is as follows:

1. *<Instruction Line 1>*.
2. *<Instruction Line 2>*.
3. ...

please generate the PDDL problem file according to the information. The PDDL problem file should have the following format: *<PDDL Problem Template>*.

LLM: Sure, I'll translate the instructions into a PDDL problem file: *<PDDL Problem File of ONE-OF-YOUR-TASK-INSTANCE>*.