

AUTOPLANBENCH: Automatically generating benchmarks for LLM planners from PDDL

Katharina Stein and Alexander Koller

Department of Language Science and Technology
Saarland Informatics Campus

Saarland University, Saarbrücken, Germany

{kstein, koller}@coli.uni-saarland.de

Abstract

LLMs are being increasingly used for planning-style tasks, but their capabilities for planning and reasoning are poorly understood. We present a novel method for automatically converting planning benchmarks written in PDDL into textual descriptions and offer a benchmark dataset created with our method. We show that while the best LLM planners do well on many planning tasks, others remain out of reach of current methods.

1 Introduction

Large Language Models (LLMs) have led to huge improvements on a large variety of NLP tasks. One active research direction concerns the question whether LLMs can do reasoning and, more specifically, planning. This encompasses both symbolic planning tasks, e.g. in robot control (Ahn et al., 2022), text-based games (Yao et al., 2022) or Minecraft problem solving (Wang et al., 2023; Zhu et al., 2023), but also less structured tasks such as question answering (e.g. Wei et al., 2022), visual programming (Gupta and Kembhavi, 2023), and the orchestration of API calls (Prasad et al., 2023).

The success of these methods on their respective benchmarks raises the question whether LLMs are actually effective at combinatorially complex tasks such as planning and reasoning or whether existing benchmarks only exercise a simple fragment of the full reasoning task – after all, LLMs give no correctness guarantees, and they do not perform any search, which was previously thought to be crucial for such tasks. Valmeekam et al. (2023a) investigated this question by manually converting two benchmark tasks for classical AI planners, which are specified in the Planning Domain Description Language (PDDL) (Ghallab et al., 1998), into natural-language tasks for LLMs. They found that LLMs were not reliably able to solve traditional planning problems, such as Blocksworld,

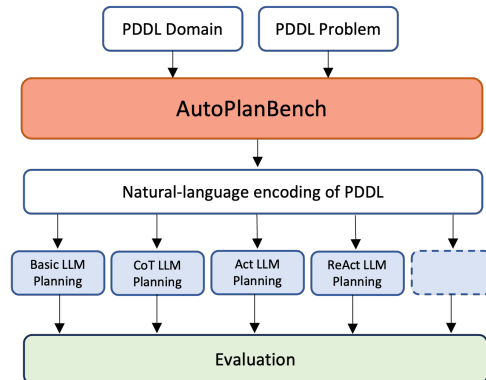


Figure 1: AUTOPLANBENCH facilitates evaluation of different LLM planners by automatically converting PDDL problems.

due to complexities of the planning problem such as subgoal interaction.

In this paper, we go beyond Valmeekam’s work by offering AUTOPLANBENCH, a method for automatically converting PDDL problem specifications into benchmarks for LLMs (Fig. 1). In contrast to Valmeekam’s approach, AUTOPLANBENCH requires no manual effort for running LLM planning on new domains; the PDDL specification is converted into natural language automatically. This makes it possible, for the first time, to investigate the planning capabilities of LLMs at a larger scale, and it ensures that knowledge that the human problem converter might have about the planning domain doesn’t accidentally slip into the natural language task. In addition to the code for converting PDDL into language, we are also releasing an initial set of 9 natural-language conversions of complex PDDL problems.

In order to convert PDDL to text at wide coverage, we use a domain-independent LLM prompting method (§3). The key technical challenge is to ensure that the (syntactic and semantic) relationship between an action and its arguments are captured correctly and that PDDL object types are made explicit in the action descriptions in an adequate

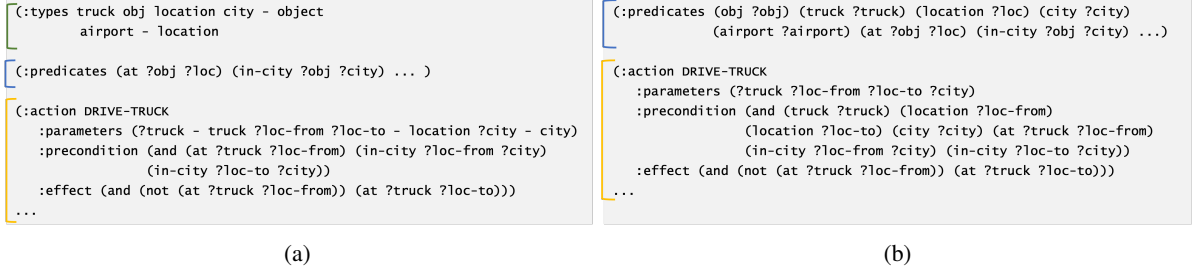


Figure 2: Excerpt from the PDDL domain definition of the Logistics domain including a subset of the state predicates (blue) and the definition of the “drive-truck” action (orange). Figure 2a shows the typed version including the definition of the type hierarchy at the top (green) and 2b the untyped version.

way. We then establish that on the three domains that Valmeekam et al. (2023b) converted by hand, our automatic method yields comparable results (§4). Finally, we demonstrate the usefulness of our wider-coverage approach by converting 9 further PDDL domains from the planning literature and evaluating a number of LLM planning approaches on them. We find that ReAct backed by GPT-4 (OpenAI, 2023) outperforms the other methods, but no LLM planning approach does well on all PDDL domains. We conclude with a discussion of the characteristics that make a planning problem difficult to solve for an LLM planner (§5).

2 Background

2.1 PDDL

The Planning Domain Description Language (PDDL) (Ghallab et al., 1998) is a popular language to encode classical planning problems. PDDL planning problems consist of a domain file and a file for each specific problem. The domain file encodes the definition of the world model including including the actions that can be executed in the world and the predicates defining the possible world states.

Figure 2a presents an excerpt of the Logistics domain. The general task in the logistics planning problems is to transport packages with trucks between locations in a city and with planes between different cities. At the top of the shown excerpt, all possible types that objects in this domain can have are stated and the type hierarchy is defined. For example, all objects that are airports are also locations and every object is of type object.

The action DRIVE-TRUCK at the bottom has four parameters “?truck”, “?loc-from”, “loc-to” and “?city” that are variables serving as placeholders for actual objects of the specified type. The parameters correspond to the truck that is driven, the start and the destination location and the city in

which the locations are. The preconditions specify that the truck can only be driven from the ?loc-from location to the ?loc-to location within the city city? if the truck is located at the ?loc-from location and both locations are in that city. The effects of the action define how executing this action changes the world state. For example, driving the truck from the start to the destination location has the effect that the predicate (?at ?truck ?loc-from) is no longer true but instead (at ?truck loc-to) becomes true.

Figure 2b also illustrates an excerpt from the logistics domain but without the typing feature, i.e. it does not include type definitions. Instead, unary predicates, such as (truck ?truck), are used in the action preconditions to ensure that actions are only executed with the appropriate kinds of objects.

A problem specification for a specific problem instance of a domain consists of a list of all available objects, the initial state and the description of the goal. The initial state is defined in terms of all state predicates that are true at the beginning of the task and the goal corresponds to a set of conditions about the world state that need to become true. A plan for the problem corresponds to a sequence of actions that when being executed one after the other starting in the initial state will result in a world state satisfying the goal conditions.

There exist different versions of the PDDL language that vary with respect to their expressiveness. In the current work, we focus on domains that contain only “and” and “not” as operators and do not include action costs or numeric values. Our approach considers both typed and untyped domains.

2.2 PDDL and LLMs

Valmeekam et al. (2023a) presented Planbench, a benchmark framework for assessing different aspects of the reasoning capabilities of LLMs based on classical planning problems formulated in

tions and effects, to the model. The preconditions and effects are presented in natural language sentences that are composed by conjoining the previously generated NL encodings for the predicates. The lower part of Table 1 presents the input and generated output for the “drive-truck” action. Providing the preconditions and effects of an action instead of providing only the action name and its parameters improves the output quality as it helps the LLM to avoid mixing up the roles of the individual parameters. For example, ?locA and ?locB would be equally good variable names as ?loc-from and ?loc-to in PDDL but would not allow any inference about which of the locations is the start and which is the destination without the additional information. Figure 8 in Appendix A.3 presents the complete prompt for the LLM.

The action example in Table 1 illustrates two interesting characteristics of the generated NL encodings. First, the order of the arguments in the generated NL description can deviate from the order of the parameters in PDDL. In PDDL the order of parameters can be arbitrary and might not match a natural sounding order of arguments in natural language. We therefore include one such PDDL-NL pair in the few-shot examples in order to prompt the LLM to generate the more appropriate ordering instead of sticking to the parameter order of the input.

Second, the NL description of “drive-truck” states the types for each of the parameters which indicates that the LLM makes use of the information in the preconditions, e.g. “?loc-from is a location”, to infer and add the types. In order to get the same behavior for typed domains, we remove the type constraints from the parameter specification and add them to the preconditions in the form of “?VAR is a TYPE” for each parameter ?VAR and its type TYPE.

3.2 Generating LLM Planning Input Files

The NL descriptions of the PDDL domains are derived based on the generated templates and include all the information from the PDDL domains. Figure 3 shows an excerpt from the NL logistics domain description. All NL descriptions start with the descriptions of the possible actions (1), followed by all preconditions (2) and then by all effects (3). If the domain is typed, we also add a verbalization of the type hierarchy at the end (4). The preconditions and effects are presented in complete sentences.

```

I can carry out the following actions:
1 drive a truck A from a location B to a location C in a city D
  fly an airplane A from an airport B to an airport C
  [...]

I have the following restrictions on my actions:
2 I can only drive a truck A from a location B to a location C in
  a city D if it is the case that C is a location and A is at B
  and D is a city and C is in the city D and B is a location and A
  is a truck and B is in the city D
  [...]

The actions have the following effects on the state:
3 Once I drive a truck A from a location B to a location C in a
  city D, it is the case that A is at C
  Once I drive a truck A from a location B to a location C in a
  city D, it is not the case anymore that A is at B
  [...]

4 Everything that is a location is also an object
  Everything that is an airport is also a location
  [...]

```

Figure 3: Excerpt from the generated NL description of the typed logistics domain, consisting of the available actions (1), their preconditions (2) and effects (3) as well as the type hierarchy (4).

For each action, its preconditions and effects are described in two sentences each: one stating all the positive preconditions or add effects and one stating all negated preconditions or delete effects respectively. This structure strikes a balance between potentially very long sentences when putting the positive and negated predicates in the same sentence and increasing the domain description by adding complete sentences for each individual precondition and effect.

In addition to the NL domain descriptions, also the NL encodings of the actual planning problems are needed. As described in Section 2, PDDL problem files comprise three parts: the available objects, the fully described initial state and a partial description of the goals state, namely the state predicates that need to be true in the end to reach the goal. Following Valmeekam et al. (2023a), we derive the NL encodings for the world states based on the predicates that define them.

The object names in PDDL problems can be any sequence of strings and they often consist of single letters and numbers as classical planning tools are not affected by the specific naming of objects - or actions and predicates. However, Valmeekam et al. (2023b) found that names can have a large effect on the planning performance of LLMs. Valmeekam et al. (2023a) convert the original object names into better suited, semantically related names, such as “yellow block” or “truck_1”.

We generate object names similar to the latter one in an automatic way based on the object types.

If a domain is typed, we name each object after its (most specific) type and enumerate them. If the domain is not typed, we use the most general type “object” for all objects names. In principle it would be possible to obtain a more specific name from the static unary predicates, i.e. the predicates that are never affected by an action, such as “(truck ?truck)”. However, objects can occur with more than one of these static predicates out of which not all might be good names for an object. Therefore, we leave the task of deriving more specific object names for untyped domains for future work.

4 Experiments

We conduct a set of experiments to assess the performance of different LLM planning approaches on a variety of different classical planning domains.

4.1 Data

We select 12 PDDL domains from the classical planning literature for our experiments. In order to investigate in which way the domain encodings generated by AUTOPLANBENCH affect the LLM planning performance compared to manual written encodings we select the Blocksworld and Logistics domain and use the manual domain encodings from Valmeekam et al. (2023a). Additionally, we also select the Depot domain for comparison and get the manual descriptions from the PlanBench repository¹.

In addition to these three domains, we select nine additional domains that contain only “and” and “not” as operators and we remove the action costs from the domains. Seven out of the 12 domains are typed. More information about the domains can be found in Table 5 in Appendix A.2. For each domain, we use the corresponding problem generators to generate problem instances and select 21 problems for each domain that fulfill the following criteria: the problem needs to be solvable and the length optimal plan is between 3 and 20. For the Blocksworld, Logistics and Depot domain we select 21 problems from the PlanBench repository with the same restrictions.

For each domain, we randomly select one of the 21 problems as few-shot example under the constraint the selected problem should have an optimal plan length of four or five steps. If none of the problems matches this criterion we select the prob-

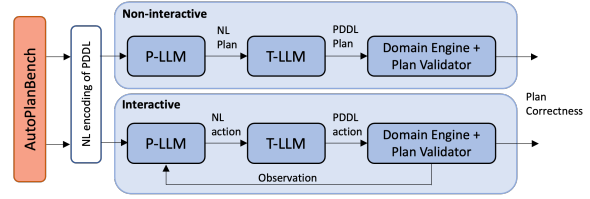


Figure 4: Overview over set-up of the non-interactive planning approaches (top) and the interactive planning approaches (bottom).

lem with the shortest optimal plan. The optimal gold plans are generated with the Fast Downward planner².

4.2 Planning Approaches

In our experiments we employ four different approaches for LLM planning: a Basic approach to generate a plan in one step, Chain-of-Thought prompting (CoT) (Wei et al., 2022) adding reasoning thoughts to the planning and two interactive approaches that incorporate observations from a simulation of the domain, namely Act, and ReAct (Yao et al., 2022). Table 2 shows the main characteristics distinguishing the four approaches.

4.2.1 Overall Set-up

Figure 4 illustrates the main process of the non-interactive and the interactive approaches. Both set-ups include two LLMs where the first one, the P-LLM generates a plan given a prompt containing the NL encoding of the domain and the problem and the second LLM (T-LLM) is responsible for translating the NL output into PDDL. In the non-interactive set-up, the P-LLM generates one complete plan that is translated step-by-step into PDDL and then passed to a domain engine that integrates the plan validator VAL³ to check the correctness.

In the interactive set-up, the P-LLM is prompted to provide only a single action at a time that is then translated into PDDL. The PDDL action is then passed to the domain engine which simulates the world model of the domain and returns the observation of applying the predicted action in the current world state. The P-LLM prompt is then extended by the predicted action and the observation and the LLM is prompted to predict the next action given the adapted input.

¹<https://github.com/karthikv792/LLMs-Planning/tree/main/plan-bench>

²<https://www.fast-downward.org/HomePage>

³<https://github.com/KCL-Planning/VAL>

	No interaction	Interaction
No Thoughts	Basic	Act
Thoughts	CoT	ReAct

Table 2: The distinguishing characteristics of the four LLM planning approaches.

Our **Basic**, non-interactive set-up is based on the set-up from Valmeekam et al. (2023a) who also let an LLM generate a complete plan given a prompt containing the NL encoding of the domain and the problem and translate the individual NL actions back into a PDDL plan that gets evaluated by VAL.

In contrast to Valmeekam et al. (2023a) we do not use a rule-based approach to translate the P-LLM outputs to PDDL but integrate the second LLM that takes care of the translation. The usage of the T-LLM makes our approach independent from specific domains and does not require any assumptions about the order of the action verb and its arguments in the NL encoding. The domain-specific few-shot examples for the T-LLM are generated entirely automatically based on the PDDL-NL pairs from the AUTOPLANBENCH step (see Figure 11) for more details). The prompts and few-shot examples for the T-LLM are identical for all planning approaches.

4.2.2 Prompts

In all approaches, the overall structure of the prompts for the P-LLM is the same (see Figure 5). All prompts start by instructing the LLM to be an assistant for providing instructions to solve small tasks. This initial instruction is followed by the description of the problem specific goal and the NL encoding of the domain.

After this the few-shot example is included, starting with the description of the goal state and the initial state followed by one few-shot example illustrating the specific planning approach. The schematic examples for the four different versions are shown in Figure 6.

Basic. The task for the LLM in the **Basic** approach is to generate one complete plan. For the few-shot examples we follow (Valmeekam et al., 2023a) and present each action in a separate line and include a special tag at the end to signal the end of the plan.

CoT. Wei et al. (2022) showed that prompting an LLM to generate a chain of thought, i.e. a sequence of reasoning steps over the task, improves

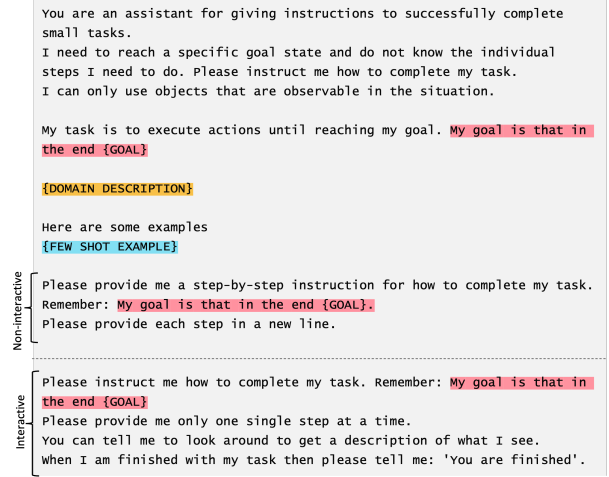


Figure 5: Structure of the prompts for the P-LLM .

the performance on a range of reasoning tasks. In our CoT approach, we let the P-LLM include such a reasoning “thought” before each of the actions in the generated plan.

ReAct. (Yao et al., 2022) introduced the ReAct prompting approach to combine chain-of-thought like reasoning with information from an environment in an interactive task. They let the model alternately output a “thought” and the next action to be executed. After generating the action the model receives an observation from the environment about the effect of the action which is then part of the model input for the next generation step. We make one small change to the original ReAct approach: we combine the reasoning and action prediction step in a single output of the LLM.

Act. In the Act approach, the P-LLM generates the plan by interacting with the environment in the same way as in the ReAct approach but does not generate reasoning thoughts for the individual actions.

All of the few-shot examples are generated automatically by translating the initial state and goal of the selected problem as well as the optimal gold plans into natural language using the NL encodings from AUTOPLANBENCH . For the interactive approaches the gold plans are passed to the domain engine to get the observations for each of the actions. In order to generate the thoughts for the ReAct examples, the few-shot examples are first created with placeholders instead of the thoughts. For the logistics and blocksworld domain we manually replace them with reasoning thoughts and then use the hand-written logistics example for prompting GPT-4 to come up with appropriate thoughts

<pre>[STATEMENT] My goal is that in the end {GOAL}. My current initial situation is as follows: {AVAILABLE OBJECTS WITH TYPES} {INITIAL STATE DESCRIPTION} [PLAN] {ACTION 1} {ACTION 2} ... [PLAN END]</pre>	<pre>My goal is that in the end {GOAL}. I: My current initial situation is as follows: {AVAILABLE OBJECTS WITH TYPES} {INITIAL STATE DESCRIPTION} You: {ACTION 1} I: {OBSERVATION FROM ENVIRONMENT} You: {ACTION 2} I: {OBSERVATION FROM ENVIRONMENT} ... You: You are finished I: Great!</pre>
<pre>[STATEMENT] My goal is that in the end {GOAL}. My current initial situation is as follows: {AVAILABLE OBJECTS WITH TYPES} {INITIAL STATE DESCRIPTION} Think: {Thought 1} Instruction: {ACTION 1} Think: {Thought 2} Instruction: {ACTION 2} ... Think: {Thought N} Instruction: You are finished [PLAN END]</pre>	<pre>My goal is that in the end {GOAL}. I: My current initial situation is as follows: {AVAILABLE OBJECTS WITH TYPES} {INITIAL STATE DESCRIPTION} You: Think: {Thought 1} Instruction: {ACTION 1} I: {OBSERVATION FROM ENVIRONMENT} You: Think: {Thought 2} Instruction: {ACTION 2} I: {OBSERVATION FROM ENVIRONMENT} ... You: Think: {Thought N} Instruction: You are finished I: Great!</pre>

Figure 6: Schematic examples of the structures of the few-shot examples for the Basic, Act, CoT and ReAct approach (left to right and top to bottom). Parts in curly brackets are placeholders for the actual, problem-specific content.

for the examples of the other domains. The Act examples are obtained by removing the observations from the ReAct ones. The costs for the interactive approaches are high due to repeatedly prompting the P-LLM with prompts of increasing length. We therefore cut the length of the ReAct (and hence also of the Act) approach to the last three steps of the problem and adapt the initial states accordingly.

The last part of the prompt consists of the instructions whether a complete plan should be provided or only the next single step and repeats the goal descriptions. In the interactive approaches, the model is additionally instructed to output “You are finished” when it predicts to have reached the goal state. If the goal state is not yet reached the planning continues otherwise it ends.

4.2.3 Observations from the Domain Engine

Taking inspiration from text-based games such as ScienceWorld (Wang et al., 2022) and AlfWorld (Shridhar et al., 2021) that have already been used in the context of LLM planning (e.g. Yao et al., 2022), we implemented a domain engine that simulates the world model of the PDDL domain by keeping track of the current state and updating it according to an input action. It makes use of the plan validator VAL to determine whether a predicted action is applicable in the current state and - if not - to obtain the information about the unsatisfied preconditions. Additionally, VAL checks whether applying the action leads to the goal state and which goal facts are not satisfied yet.

Based on the information provided by VAL, the domain engine generates an observation for the

P-LLM. If the action is executable, this observation corresponds to the statement about executing this action. For example, if the predicted action is “Drive truck truck_0 from location location_0 in city city_0 to location location_1 in the same city” the observation would be “I drive truck truck_0 from location location_0 in city city_0 to location location_1 in the same city.”.

If the predicted action is not executable the observation informs about this and states which preconditions are not satisfied. For example, if the truck is at the wrong location the observation observation would be “I cannot drive truck truck_0 from location location_0 in city city_0 to location location_1 in the same city because truck_0 is not at location_0.”.

In the P-LLM in the interactive approach erroneously predicts “You are finished” the observation from the environment states that the goal state is not yet reached and lists the goal facts that are not satisfied as an explanation.

4.3 Results

In this section we report the results of running the four LLM planning approaches on the 12 planning domains encoded by AUTOPLANBENCH. For the P-LLM and the T-LLM we use the GPT-4 model from OpenAI via their API (see Appendix A.1).

Metrics. We calculate the accuracy and the average length factor of the correct predicted plans compared to the optimal plans (LF). A plan is considered to be correct if the goal state is reached and the P-LLM correctly predicts to be finished in the goal state. Under this definition, we also consider an interactively generated plan as correct if not all of the predicted actions are executable as long as the LLM is able to correct its mistakes and to finally reach the goal. For the LF computation we only count the executable predicted actions.

In order to assess how many of the plans generated interactively would have been correct also under the stricter constraint that the LLM is not allowed to make a mistake we also report the proportion of problems solved without any mistakes out of all problems (Acc^0).

In the interactive approaches the planning process is not guaranteed to stop because the stopping condition is that the P-LLM predicts to be in the goal state when being in the goal state. However, it is possible that the P-LLM gets stuck predicting non-executable actions or ends up in a dead end

		Blocksworld		Logistics		Depot	
		APB	Manual	APB	Manual	APB	Manual
Basic	Acc	0.50	0.45	0.10	0.15	0.05	0.05
	LF	1.18	1.15	1.00	1.00	1.06	1.22
CoT	Acc	0.30	0.55	0.3	0.3	0.15	0.05
	LF	2.80	4.03	2.89	3.18	2.69	2.36
Act	Acc	0.80	0.90	0.55	0.75	0.20	0.20
	Acc ⁰	0.25	0.5	0.20	0.20	0.05	0.05
	LF	1.39	2.12	1.70	1.60	1.13	1.14
ReAct	Acc	0.95	0.90	0.70	0.75	0.4	0.35
	Acc ⁰	0.55	0.75	0.35	0.45	0.10	0.00
	LF	1.42	1.67	1.19	1.11	1.14	1.22

Table 3: Results of running the different LLM planning approaches using GPT 4 on the blocksworld, logistics and depot domain when using the NL domain encodings generated by AUTOPLANBENCH (APB) and the domain encodings from Valmeekam et al. (2023a) (Manual).

	N	Basic		CoT		Act			ReAct		
		Acc	LF	Acc	LF	Acc	Acc ⁰	LF	Acc	Acc ⁰	LF
Ferry	20	0.10	1.0	0.95	3.68	0.40	0.10	2.42	1.00	1.00	1.14
Floortile	20	0.00	–	0.00	–	0.00	0.00	–	0.00	0.00	–
Goldminer	20	0.10	1.0	0.2	3.25	0.30	0.1	1.0	0.4	0.3	1.0
Grid	20	0.15	1.0	0.2	3.40	0.70	0.25	1.42	0.8	0.25	1.15
Grippers	20	0.40	1.80	0.86	3.99	0.55	0.3	1.71	0.95	0.95	1.37
Movie Seq	20	0.60	1.0	0.00	–	1.0	0.45	1.37	0.85	0.0	1.74
Rovers	20	0.00	–	0.10	3.28	0.50	0.00	1.18	0.55	0.05	1.11
Satellite	20	0.10	1.00	0.50	3.35	0.90	0.20	1.46	0.9	0.55	1.23
Visitall	20	0.90	1.04	0.85	3.36	1.00	0.90	1.08	1.00	0.90	1.10

Table 4: Results of running the different LLM planning approaches using GPT-4 on the AI planning domains using the domain encodings generated by AUTOPLANBENCH.

state. In order to limit the run time and costs we set the maximum number of prediction steps to 24 for our experiments and leave it for future work to determine how to balance costs and the number of possible steps. We find that none of the incorrect plans generated by `Act` and `ReAct` reaches the step limit without having made a mistake. Therefore the Acc^0 values are not affected by the length restriction.

Table 3 shows the LLM planning results on the Blocksworld, Logistics and Depot domain when providing the NL encodings generated by AUTOPLANBENCH (APB) compared to when providing the manual PlanBench encodings. For several of the combinations of domain and planning approach we observe comparable planning performance for the two NL encodings while for others we observe a drop in performance when using the APB encodings that we will discuss in more detail in the next section.

Table 4 presents the planning results on the nine additional domains using the NL encodings generated by AUTOPLANBENCH. Overall, the results in Table 3 and 4 illustrate that predicting correct plans without mistakes is a difficult task for LLMs. For many of the domains, the Acc of the `Basic` approach and the Acc^0 of the `Act` are very low. Adding reasoning thoughts improves the performance for most of the domains and unsurprisingly the `ReAct` LLM planning yields the best results in terms of Acc . One exception to this is the `MovieSeq` domain which will be discussed in more detail in §5.

Turning to the effect of allowing the LLM planner to interact with the environment we find that the Acc of the `Act` and `ReAct` approaches are always better than the `Basic` and `CoT` results respectively.

Comparing the performances between the 12 different domains we see large differences in the planning performance: Some domains such as `Floortile`, `Goldminer` and `Rovers` are particularly difficult for the LLMs while the `Visitall` domain is solved almost perfectly even with the `Basic` planner.

Looking at the LF metric we observe that if the `Basic` approach finds a correct plan it is close to optimal on average. In contrast, the `CoT` approach generates much longer correct plans.

5 Discussion

In this section we discuss the results from the planning experiments in more detail in order to get a better understanding of what makes a planning problem or domain difficult for LLM planning.

5.1 AUTOPLANBENCH Encodings

The main contribution of our work is the AUTOPLANBENCH tool to automatically generate NL encodings of PDDL domains with the goal to use them in the context of LLM planning and research on the capabilities of LLMs to reason. Creating NL encodings of PDDL domains is costly and can be hard and time-consuming without additional knowledge about the domain. Therefore, our automatic approach allows to do LLM planning on classical planning domains at a larger scale and does not require and domain expertise.

When comparing how well different LLM planners perform when provided the automatic encodings compared to when providing the manual ones from Valmeekam et al. (2023a) we see some drop in performance for some of the experiments on Blocksworld and Logistics. Looking in more detail at the two versions of encodings we identify two main differences: 1) the manual encodings contain additional information not or not explicitly expressed in the PDDL and 2) the objects are named in different ways.

For example, the manual Blocksworld encodings states that “A block is clear if the block has no other blocks on top of it and if the block is not picked up”. In the PDDL domain and hence in the ABP encodings this information is not available. In the manual Logistics encoding, explicitly states that the task is “to transport packages within cities via trucks and between cities via airplanes”. And while this information is part of the PDDL domain it needs to be inferred from the preconditions of the actions and is not available in an explicit way. Additionally, the manual encoding includes the information that there is exactly one truck and on airport in each city. However, this is not even part of the PDDL domain itself but meta knowledge about the domain and conventions for the generated problems.

The second main difference between the NL encoding versions for Blocksworld and Logistics concerns the naming of the objects. As both domains are not typed AUTOPLANBENCH assigns to all objects names of the form “object_x”. In contrast, the manual logistics encodings map each object

to a name consisting of its type. For example, an object that is a truck would be called “truck_0” in the manual encodings but “object_0” in the automatic NL encodings if not types are available. The manual Blocksworld encodings include even more elaborate object names such as “yellow block” and “red block” following the convention to refer to the blocks by color although this is not part of the PDDL definitions.

The manual depot domain also includes external domain information but as the domain is typed there is no difference between the object namings and the results for this domain are comparable for both encoding versions.

Overall, we therefore conclude that any gains in the LLM planning performance when using the manual encodings from Valmeekam et al. (2023a) are most likely caused by additional knowledge that the annotators had about the domain and that was added in to the NL encodings on top of the actual domain definition. This indicates that generating NL encodings of PDDL domains manually is not better *per se*. Instead, what makes planning with manual encodings slightly easier for LLMs is the amount of external domain knowledge that an annotator can or decides to provide to the model for a specific domain. In contrast, automatically generated encodings allow more systematic and fairer comparisons.

5.2 Challenges for LLM Planning

Overall, our results support previously made claims that LLMs do not excel at the generation of complete plans from scratch for classical planning problems but yield very low results for many domains (e.g. Valmeekam et al., 2023a; Liu et al., 2023). In line with previous work on prompting approaches for reasoning we find that letting the model reason about the task and next steps improves the performance of the LLMs on planning.

Unsurprisingly, allowing the LLM planner to interact with the environment and to continue after predicting a non-executable action leads to a higher success rate of reaching the goal state. On the one hand it might not be feasible in many real world tasks to generate a plan in this way as the costs are higher and when trying to execute wrong predicted actions might have bad effects in real tasks. On the other hand, the increase between Acc^0 and Acc for the interactive LLM planners illustrates that LLMs are capable of recovering and

learning from mistakes when being given informative feedback. This is still a desirable ability as not all environments and their rules are fully observable and unforeseen situation can occur.

As stated in Section §4 there are large differences in the performance of the LLM planners across domains. The easiest domain seems to be Visitall. In this domain the only task is to have visited a number of specific locations in a grid world and the only precondition for moving from one location to another is that the two locations need to be connected. Additionally, the average of the optimal plan lengths is the lowest for the Visitall problems.

The most difficult domains where even ReAct planning improves the results only to a limited extent are the Depot, Floortile, Goldminer and Rovers domain. These are also the four domains with the largest average plan length of the problems. These observations indicate that problems that require longer plans are more difficult. One reason could be that LLMs are bad at planning long-term interactions or the problem might be that they do not perform well on generalizing from shorter few-shot examples to longer instances. However, plan length alone cannot explain the performance differences as for example the average plan length of the Grippers problems is slightly higher than the one of the Grid problems which are solved less often.

Similarly, we observe some relation between the planning accuracy and the number of predicates defining the initial state. The number of predicates in the initial state indicates how complex the world states in the domain can be and domains with an higher average tend to be more difficult. Table 6 in Appendix A.2 provides an overview over characteristics of the generated problems of all domains.

Regarding the Floortile domain, we replicate the findings from Liu et al. (2023) who also observed that this domain is not solvable when letting GPT-4 generate a complete plan. We additionally show that for this domain even more advanced planning approaches fail. There are two characteristics of this domain that can explain the complexity: First, the domain includes complex spatial relations as the task is to move in a grid world and for all pairs of adjacent locations it is specified whether they are in and “left”, “up”, “right” or “down” relation. Additionally, a different move action is required for each direction. This complexity has also been pointed out by Liu et al. (2023). Another special characteristic is that there can be dead states be-

cause the robots need to paint tiles but cannot move on tiles already painted. Therefore, they can end up in a state without any options left for solving the task.

Lastly, we briefly discuss the results on the Ferry and Movie Seq domains which do not fit into the result patterns of the other domains. For the Ferry domain we find that the predicate “(at-ferry ?l)” gets translated into “?l is at the ferry” although it should be “the ferry is at ?l”. This illustrates the complexity of correctly extracting the semantic relations between the parameters and objects or actions if the PDDL names intuitively suggest a different interpretation. The generated ReAct and CoT examples “fix” this mistake in the reasoning steps hence improving over the approaches without thoughts by much larger magnitude than for the other domains.

The results of the Movie Seq domain are interesting because adding the reasoning thoughts leads to a worse performance. In this domain, the task is to buy different kinds of snacks, start the movie, rewind the movie and reset the counter to zero. Starting the movie needs to happen before the rewinding which needs to be done before setting the counter. In order to buy the snacks the counter needs to be reset but the movie must not be rewound. Therefore, an optimal plan is to buy all snacks at the beginning. If this is not the case, the rewinding needs to be undone first and afterwards the sequence of starting the movie, rewinding and resetting needs to be repeated. In the ReAct and CoT approach the few-shot examples are limited to problems with an optimal length of three. Therefore, the LLMs only get an example of a problem where the snacks have already been bought and the first action is starting the movie. A manual inspection of the predicted plans suggests that the model is not able to correctly understand the preconditions for buying the snacks and planning the actions accordingly.

These observations relate to the broader research question of how the selection of few-shot examples affects the performance of LLMs on a task. Valmeekam et al. (2023b) also report differences in the LLM planning accuracy when changing the examples plans provided. Levy et al. (2023) propose a method to select diverse few-shot examples to improve LLM performance on tasks involving compositionality. Adopting such an approach for LLM planning tasks is an interesting direction for

future investigations.

6 Conclusion

The main contribution of this work is the AUTOPLANBENCH approach for automatically converting classical planning benchmarks from the formal planning language PDDL into natural language descriptions. Our proposed method enables testing different LLM planning approaches on classical planning domains at large scale without requiring manual effort and domain experts. While AUTOPLANBENCH currently supports only PDDL of limited expressiveness the approach could be extended to include more complex operators in the future.

Our framework currently provides four different planning approaches based on different common prompting strategies. While our experiments focus on comparing different LLM planning approaches we hypothesize that our set of automatically generated NL encodings can be a valuable resource to be used in other approaches investigating the usage of LLMs for planning tasks, including the fine-grained assessment tasks from PlanBench (Valmeekam et al., 2023a). We provide an initial analysis of what makes specific problems and domains difficult for LLM planners and identify a number of potential characteristics affecting the LLM planning but further research is required to get a better understanding of the challenges for LLM planning approaches.

7 Acknowledgements

We thank Julia Wichlacz, Jörg Hoffmann and Dan Fiser for fruitful discussions and their valuable feedback.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. 2022. *Do as i can,*

- not as i say: Grounding language in robotic affordances.
- Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manueal Veloso, Daniel Weld, and David Wilkins. 1998. Pddl - the planning domain definition language.
- Tanmay Gupta and Aniruddha Kembhavi. 2023. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14953–14962.
- Itay Levy, Ben Bogin, and Jonathan Berant. 2023. Diverse demonstrations improve in-context compositional generalization.
- Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.
- OpenAI. 2023. [Gpt-4 technical report](#).
- Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2023. [Adapt: As-needed decomposition and planning with language models](#).
- Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. [Alfworld: Aligning text and embodied environments for interactive learning](#).
- Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2023a. [Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change](#). In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
- Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023b. [On the planning abilities of large language models – a critical investigation](#).
- Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. [Scienceworld: Is your agent smarter than a 5th grader?](#)
- Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023. [Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents](#).
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. [Chain-of-thought prompting elicits reasoning in large language models](#). In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.
- Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. 2023. [Translating natural language to planning goals with large-language models](#).
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. [React: Synergizing reasoning and acting in language models](#). *arXiv preprint arXiv:2210.03629*.
- Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, Yu Qiao, Zhaoxiang Zhang, and Jifeng Dai. 2023. [Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory](#).

A Appendix

A.1 Experimental Set-up: Model Parameters

All experiments with GPT-4 were conducted using the OpenAI API between 9th and 14th of November 2023. For all models and tasks we set the temperature to 0.0 and use caching to reduce the cost: If the model has already received the same input (including the prompt, the user message and the history) before, the previously generated response text is retrieved from the cache. This affects in the first place the T-LLM because all planning approaches and individual problems have different initial prompts and therefore the P-LLM should never get exactly the same input twice.

APB-LLM . We use GPT-4 for converting the PDDL predicates and actions into natural language templates with the number of maximum tokens to generate set to 50 and keep the default values for the other parameters.

P-LLM . We use the GPT-4 model to generate the plans. We did not limit the maximum number of output tokens and kept the default values for all parameters.

T-LLM . For the translations or the natural language instructions to PDDL we use GPT-4 with the number of maximum tokens to generate set to 256 and keep the default values for the other parameters.

A.2 PDDL Domains and Problems

Table 5 provides an overview over the tasks of the different planning domains included in our experiments. Table 6 presents for the 21 problems of each domain the minimum, maximum and average optimal plan length, number of facts in the goal definition, number of predicates in the initial state and the number of available objects.

A.3 Prompts and Few-shot Examples

In this section, additional examples and templates of the prompts and few-shot examples used are provided.

Figure 7 and 8 show the prompts used to derive the core natural language encodings for the predicates and the actions based on which the overall domain encodings are generated. Figure 10 and 9 present the prompt templates for the interactive and non-interactive planning approaches. In Figure 11 an example for the prompt for the T-LLM in the Blocksworld domain is presented. The Figures 12 and 13 present the ReAct few-shot examples that

we manually created for the logistics domain encoded by ours and by Valmeekam et al. (2023a)’s encodings. The example in Figure 12 was used to generate the ReAct few-shot examples for all domains except the Blocksworld domain.

Domain	Typed	Description
Blocksworld	no	The task is to rearrange to a set of blocks into specified stacks
Depot	yes	The task is to move crates between depots and distributors using a truck. In order to load and unload the crates a hoist at the specific place must be used. All depots and distributors are connected.
Ferry	no	The task is to transport cars using a ferry to their goal locations. The ferry can only transport one car at a time and only move between non-equal locations. Non-equality is specified by state predicate for all pairs of locations
Floortile	yes	A set of robots need to paint floortiles in a grid world using two colors. Robots can only paint floortiles that are located up or down the current position. There are four actions for moving, one for each direction and robots cannot move on painted tiles.
Goldminer	yes	A robot needs to navigate a grid world to reach the location with gold. The locations are blocked by soft or hard rocks that need to be removed using a laser or a bomb (only soft rocks). Using the laser on the gold location destroys the gold. Connectivity of locations is defined by a predicate.
Grid	no	A robot needs to move in a grid world and move different types of keys to specific locations. Locations can be locked and a matching key can be used to open them. Connectivity of locations is defined by a predicate.
Grippers	yes	A number of robots with two gripper hands need to transport balls between rooms. All rooms are connected.
Logistics	no	The task is to transport packages either within cities using trucks or between cities using airplanes. All cities and locations are connected.
Movie Seq	no	Adapted from the Movie domain where the task is to buy one of each of 5 kinds of snacks, rewind the movie and reset the counter to zero. The rewinding needs to happen before setting the counter but buying snack can happen at any state. In the adapted domain the snacks need to be bought before rewinding the movie and resetting the counter. If snacks are still missing afterwards the rewind action needs to undone before buying again.
Rovers	yes	A number of rovers need to navigate between waypoints, find substance samples, calibrate cameras, take pictures and communicate with a lander
Satellite	yes	The task is to let satellites take images of observations in specific modes. This involves turning on the instruments supporting the target mode, calibrate the satellite and instruments, adapting the direction and taking the images.
Visitall	yes	A robot needs to move in a grid world until having visited all specified goal locations. The robot can only move to connected locations and some locations in the grid are not available.

Table 5: Overview over the nine different planning domains.

Domain	Plan length			N Goal Facts			N Initial Facts			Objects		
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
Blocksworld	2	12	7.05	1	3	2.10	6	8	6.90	4	4	4.00
Depot	5	16	12.24	2	3	2.71	25	25	25.00	18	18	18.00
Ferry	4	16	8.95	2	6	4.62	12	30	20.76	5	10	7.67
Floortile	8	19	14.10	4	8	5.76	24	57	38.57	10	19	13.90
Goldminer	5	19	10.62	1	1	1.0	17	51	33.52	4	12	8.14
Grid	3	20	8.14	1	4	1.33	23	89	54.71	6	20	13.29
Grippers	3	18	9.24	3	7	4.57	6	14	9.71	9	19	13.43
Logistics	3	18	8.43	1	4	1.95	18	28	22.76	9	14	11.38
Movie Seq	4	8	6.48	3	37	5.48	6	19	11.10	5	18	10.10
Rovers	6	16	10.43	2	5	3.14	25	46	35.05	11	16	13.19
Satellite	4	14	7.81	1	5	2.76	5	25	12.43	6	14	10.67
Visitall	3	11	4.57	3	11	4.33	8	28	14.00	4	11	6.0

Table 6: Overview over the optimal plan lengths, number of goal conditions, predicates in the initial state and the number of objects for the 21 problems used in the experiments.

few-shot examples	Your task is to generate templates for natural language descriptions for actions and predicates defined in the PDDL planning language. Tokens starting with '?' are variables and serve as placeholders. They should always be surrounded by curly brackets in your output. For actions, also include the type for each parameter in the brackets if this information is available. Do not start your output with a capitalized word and do not include sentence final punctuation.	
	Original: (obj ?obj)	Output: {?obj} is an object
	Original: (planet ?ob)	Output: {?ob} is a planet
	Original: (in ?obj ?loc)	Output: {?obj} is in {?loc}
	Original: (in-cottage ?obj ?cott)	Output: {?obj} is in the cottage {?cott}
	Original: (undertree ?obj)	Output: {?obj} is under the tree
	Original: (handempty)	Output: the hand is empty
	Original: (washing ?car)	Output: {?car} is being washed
	Original: {PDDL PREDICATE}	Output:

Figure 7: The prompt for the AUTOPLANBENCH LLM to generate natural language encodings of the PDDL predicates

few-shot examples

Your task is to generate templates for natural language descriptions for actions and predicates defined in the PDDL planning language. Tokens starting with '?' are variables and serve as placeholders. They should always be surrounded by curly brackets in your output. For actions, also include the type for each parameter in the brackets if this information is available. Do not start your output with a capitalized word and do not include sentence final punctuation.

Original: action: grasp
parameters: (?obj ?obj2)
preconditions of grasp: ?obj is a hand and ?obj2 is a box and ?obj is empty and ?obj2 is on the table
effects of grasp: it becomes true that ?obj2 is being held by ?obj and it is not the case anymore that ?obj is empty and ?obj2 is on the table
Output: grasp {box ?obj2} with {hand ?obj}

Original: action: RIDE-HORSE
parameters: (?h ?fl ?tl)
preconditions of RIDE-HORSE: ?h is a horse and ?fl is a location and ?tl is a location and ?h is at ?fl
effects of RIDE-HORSE: it becomes true that ?h is at ?tl and it is not the case anymore that ?h is at ?fl
Output: ride {horse ?h} from {location ?fl} to {location ?tl}

Original: action: put-down
parameters: (?obj)
preconditions of put-down: ?obj is an object and ?obj is being held
effects of put-down: it becomes true that ?obj is clear and hand is empty and ?obj is on the table and it is not the case anymore that ?obj is being held
Output: put down {object ?obj}

Original: action: move
parameters: (?d ?o ?s)
preconditions of move: ?d is a room and ?o is a chair and ?s is a room and ?o is in ?s
effects of move: it becomes true that ?o is in ?d and it is not the case anymore that ?o is in ?s
Output: move {chair ?o} from {room ?s} to {room ?d}

Original: {PDDL ACTION + PARAMETERS & NL PRECONDITION + EFFECTS}
Output:

Figure 8: The prompt for the AUTOPLANBENCH LLM to generate natural language encodings of the PDDL actions

You are an assistant for giving instructions to successfully complete small tasks.
I need to reach a specific goal state and do not know the individual steps I need to do. Please instruct me how to complete my task.
I can only use objects that are observable in the situation.

My task is to execute actions until reaching my goal. My goal is that in the end {GOAL}

I can carry out the following actions:
{ACTIONS}

I have the following restrictions on my actions:
{PRECONDITIONS}

The actions have the following effects on the state:
{EFFECTS}

{TYPE HIERARCHY}

Here are some examples
{FEW SHOT EXAMPLE}

Please provide me a step-by-step instruction for how to complete my task.
Remember: My goal is that in the end {GOAL}.
Please provide each step in a new line.

NL domain encoding

Figure 9: Prompting template used for the noninteractive planning approaches. {GOAL} corresponds to the problem specific goal description, {ACTIONS}, {PRECONDITIONS}, {EFFECTS} and {TYPE HIERARCHY} to the domain specific descriptions and the FEW SHOT EXAMPLE is the approach specific planning example.

You are an assistant for giving instructions to successfully complete small tasks.

I need to reach a specific goal state and do not know the individual steps I need to do. Please instruct me how to complete my task.

I can only use objects that are observable in the situation.

My task is to execute actions until reaching my goal. My goal is that in the end {GOAL}

I can carry out the following actions:
{ACTIONS}

I have the following restrictions on my actions:
{PRECONDITIONS}

The actions have the following effects on the state:
{EFFECTS}

{TYPE HIERARCHY}

Here is an example of one complete round of providing me instructions.
{FEW SHOT EXAMPLE}

Please instruct me how to complete my task. Remember: My goal is that in the end {GOAL}

Please provide me only one single step at a time.

You can tell me to look around to get a description of what I see.

When I am finished with my task then please tell me: 'You are finished'.


A yellow bracket on the right side of the text blocks, spanning from the actions section down to the type hierarchy section, with the label "NL domain encoding" next to it.

Figure 10: Prompting template used for the interactive planning approaches. {GOAL} corresponds to the problem specific goal description, {ACTIONS}, {PRECONDITIONS}, {EFFECTS} and {TYPE HIERARCHY} to the domain specific descriptions and the FEW SHOT EXAMPLE is the approach specific planning example.

Your task is to translate natural language instructions into a schematic planning language.

Each of the instructions should be converted to a tuple that matches any of the example templates below, where the tokens starting with a question mark should be replaced by the actual object.

If none of the action templates below matches the original instruction then choose one with a similar meaning if possible. Otherwise, keep the original one.

These are the action templates and their descriptions:

(pick-up ?obj)
description: pick up object {?ob} from the table
...

If one of the following objects matches the objects in the original instruction then replace each of the tokens with question marks by the appropriate object. Otherwise, replace it with the original object.

These are the available objects:

object_26
rain
star_23
...
object_0
object_1
...

Here are some examples for your task:

Input: Now, pick up object object_26 from the table.
Output: (pick-up object_26)

Input: Once you are done, please stack object star_23 on top of object rain.
Output: (stack star_23 rain)

Input: First, put down object object_26 and then unstack object object_50 from object socket.
Output: (put-down object_26)
(unstack object_50 socket)

Input: The next step is to pick up object object_50 from the table.
Output: (pick-up object_50)

Input: {ACTION PREDICTED BY P-LLM}
Output:

Figure 11: The prompt for the T-LLM to translate the predicted natural language actions back into PDDL.

My goal is that in the end object_0 is at object_3.

I: My current initial situation is as follows:

There are 5 entities: object_0, object_1, object_2, object_3, object_4
 Currently, object_0 is an object, object_0 is at object_1, object_1 is a truck, object_1 is at object_2, object_2 is a location, object_2 is in the city object_4, object_3 is a location, object_3 is in the city object_4, object_4 is a city

You:

Think: The object object_0 is currently at the truck object_1 and the truck object_1 is at location object_2 in city object_4. The object object_0 needs to be moved to location object_3 in city object_4.

Instruction: drive the truck object_1 from the location object_2 to the location object_3 in the city object_4

I: I drive the truck object_1 from the location object_2 to the location object_3 in the city object_4

You:

Think: Now, the truck object_1 is at location object_3 in city object_4 and the object object_0 is still at the truck object_1.

Instruction: unload the object object_0 from the truck object_1 at the location object_3

I: I unload the object object_0 from the truck object_1 at the location object_3

You:

Think: Now, the object object_0 is at location object_3

Instruction: You are finished.

I: Great!

Figure 12: Manually created ReAct few-shot example for the Logistics domain encoded by AUTOPLANBENCH .

My goal is that in the end package_0 is at location_0_1.

I: My current initial situation is as follows:

There are 5 entities: package_0, truck_0, location_0_0, location_0_1, city_0
 Currently, package_0 is an object, package_0 is at truck_0, truck_0 is a truck, truck_0 is at location_0_0, location_0_0 is a location, location_0_0 is in the city city_0, location_0_1 is a location, location_0_1 is in the city city_0, city_0 is a city

You:

Think: The object package_0 is currently at the truck truck_0 and the truck truck_0 is at location location_0_0 in city city_0. The object package_0 needs to be moved to location location_0_1 in city city_0.

Instruction: drive truck_0 from location_0_0 to location_0_1 in city_0

I: I drive truck_0 from location_0_0 to location_0_1 in city_0

You:

Think: Now, the truck truck_0 is at location location_0_1 in city city_0 and the object package_0 is still at the truck truck_0.

Instruction: unload package_0 from truck_0 at location_0_1

I: I unload package_0 from truck_0 at location_0_1

You:

Think: Now, the object package_0 is at location location_0_1

Instruction: You are finished.

I: Great!

Figure 13: Manually created ReAct few-shot example for the Logistics domain encoded using the descriptions from Valmeekam et al. (2023a).