Automating the Generation of Prompts for LLM-based Action Choice in PDDL Planning

Katharina Stein[a,*], **Daniel Fišer**[a], **Jörg Hoffmann**[a, b] **and Alexander Koller**[a]

[a]Saarland Informatics Campus, Saarland University, Saarbrücken Germany
[b]German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany

Abstract. Large language models (LLMs) have revolutionized a large variety of NLP tasks. An active debate is to what extent they can do reasoning and planning. Prior work has assessed the latter in the specific context of combinatorial planning with PDDL input, based on manually converting three PDDL domains into LLM prompts. Here we automate this conversion step, showing how to automatically generate LLM text prompts from PDDL input. We show empirically that our automatically generated prompts result in similar planning performance as the previous manually generated ones. Beyond this, our automatic machinery enables us to run much larger experiments, providing for the first time a reasonably broad evaluation of LLM action-choice performance in PDDL. Overall, while LLM action choice at this point lags far behind symbolic planners, our results shed a somewhat more encouraging light than previously suggested. All our LLM configurations soundly beat random action choice, showing that the LLM does carry *some* information about general PDDL planning; in some domains, our best LLM configuration scales up further than a state-of-the-art optimal planner.

1 Introduction

Large language models (LLMs) have revolutionized a large variety of natural language processing tasks. A recent research trend investigates whether LLMs can also do planning. The word "planning" here is used in a broad sense, encompassing, for example, robot control [1], text-based games [27] or Minecraft problem solving [24, 28], but also less structured tasks such as question answering [e.g. 25, 12], visual programming [5], and the orchestration of API calls [16].

Here we address combinatorial planning with PDDL input [4, 7], i.e., the core focus of the AI Planning community. The use of LLMs in this context is, at this stage, still in its infancy. First works explored what form of input to provide to the LLM (PDDL, natural language) [19, 21, 22]; recent work explored the generation of program code for generalized planning [20]. Here, we follow up on the prominent work line by Valmeekam et al. [21, 22], who investigated the ability of LLMs to produce sequential plans—action sequences achieving the goal—for PDDL planning tasks. Valmeekam et al. experiment with three wide-spread benchmark domains in AI Planning, namely Blocksworld, Depots and Logistics, for each of which they manually engineer natural language descriptions of the actions and predicates. They ask the LLM to produce a plan, thus serving as a form of satisficing planner that gives no plan correctness (nor optimality) guarantee. Valmeekam et al. find that LLMs (both GPT-3.5 and GPT-4) are unable to reliably produce correct plans in their 3 benchmark domains, lagging far behind symbolic planning methods.

* Corresponding Author. Email: kstein@lst.uni-saarland.de.

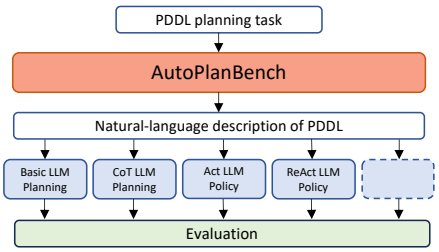


Figure 1. AUTOPLANBENCH implements our automatic conversion from PDDL to natural-language prompts, as well as four different LLM action choice mechanisms.

We extend Valmeekam et al.'s work by automating the conversion of PDDL into natural-language prompts for LLM plan generation (and, more generally, action choice, see below). We thus turn the use of LLMs with natural-language prompts into an actual automatic machinery that does not rely on any domain-dependent knowledge provided by humans apart from the PDDL itself (the standard profile of domain-independent planning methods in the AI Planning community). 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 way.

We address these challenges by leveraging LLMs themselves to support the conversion from PDDL to natural language. We first use GPT-4 [15] to convert PDDL predicates into natural language based on a few generic conversion examples. We then generate natural language descriptions of PDDL action schemas by first converting the preconditions and effects based on a simple composition of the previously converted predicates, and then providing again a few generic examples for the overall conversion step. Finally, we compose the overall domain-description prompt from the predicate and action schema descriptions. Experimenting with the same benchmarks used by Valmeekam et al., we show empirically that our automatically generated prompts result in similar plan-generation performance as the previous manually generated ones.

Beyond this, our fully automated machinery enables us to run much larger experiments than Valmeekam et al., providing for the first time a reasonably broad evaluation of LLM action-choice performance in PDDL planning based on natural language conversion. Figure 1 gives an overview of our implemented framework

AUTOPLANBENCH. We experiment with four different variants of LLM action choice mechanisms[1]:

Basic LLM Planning: Domain and problem description provided once, LLM generates a plan. Same as Valmeekam et al. [21, 22].
CoT LLM Planning: The LLM also generates a plan, however the prompt is enriched by Chain-of-Thought (CoT) prompting [25], where the LLM is prompted to generate "thoughts" between the predicted actions in the plan. Valmeekam et al. [22] experimented with a simple version of this, producing the states before and after executing each action; here we allow more flexible reasoning, e.g., about the next required actions.
Act: Here the LLM is used as an action policy instead of a plan generator, choosing an individual action for the state at each step in the plan. (A similar idea was explored by Yao et al. [27] in different, non-PDDL-planning, contexts.)
ReAct: This configuration, inspired by Yao et al. [27], uses CoT within Act, generating intermediate thoughts between the actions.

In addition to the three domains provided by Valmeekam et al., we convert 9 more PDDL domains from the planning literature. We systematically evaluate the four LLM action choice mechanisms. To assess the planning capabilities of these mechanisms, we provide comparisons to (1) LLM action choice mechanisms based directly on PDDL [22, 19] rather than natural-language encodings thereof; (2) random action selection as a sanity test to find out whether the LLM contains any information about planning at all; (3) blind breadth-first search as a trivial symbolic baseline; and (4) a state-of-the-art optimal planner and a strong satisficing planning baseline [10, 11], to assess the comparison to symbolic planners.

Overall, while LLM action choice at this point lags far behind state-of-the-art symbolic planning, our results shed a somewhat more encouraging light than previously suggested by Valmeekam et al. The comparison to (1) shows (expectedly) that natural language prompts yield superior performance. In comparison (2), all four LLM action choice mechanisms soundly beat random action choice, in case of Act and ReAct by a drastic coverage difference (5-6 times more instances solved overall). This convincingly shows that the LLM does carry *some* information about general PDDL planning, an observation that we believe is not self-evident given the training on internet text which hardly contains much information about most planning benchmark domains.[2]

The comparisons (3) and (4) to symbolic planners relying on search are less favorable, as one would expect. The satisficing planner (directly comparable as it does not provide a plan-quality guarantee) reigns supreme throughout. On the positive side, ReAct outperforms breadth-first search in two domains (Ferry and Visitall), and even outperforms the optimal planner in two domains (Ferry and Grippers). While these are isolated islands of good performance, they do show promise for LLM planning abilities, in particular as this performance is obtained without any search. We are releasing the entire AUTOPLANBENCH code base, as well as our LLM benchmark dataset comprising 12 domains, to support further research into the use of LLMs in PDDL planning. Overall, our contributions are:

Automating the conversion of PDDL to natural-language prompts. Showing that this automation does not result in a performance loss relative to the previous hand-crafted prompts.
Implementing four variants of LLM action choice in PDDL planning, transferring ideas previously proposed in different contexts.
Broad experiments on 12 domains and systematically evaluating LLM action choice against representative symbolic planners.
Publicly available code base in AUTOPLANBENCH.[3]

2 Background

The Planning Domain Description Language (PDDL) [4] is a schematic language based on first-order logic introduced for the International Planning Competitions[4] that became a standard definition language for classical planning problems. Each task in PDDL consists of a domain and problem file. The domain file defines the world model using predicates describing possible world states, and actions whose execution changes the current state. The problem file defines a specific instance from the domain by specifying available objects, the initial state and the goal.

Each action is defined by its precondition specifying what has to be true in the state where the action is applied, and by its effect saying what will become true (add effect) and false (delete effect) in the resulting state after action's execution. The solution to a planning problem is a plan—a sequence of actions leading from the initial state to a state where the goal condition holds.

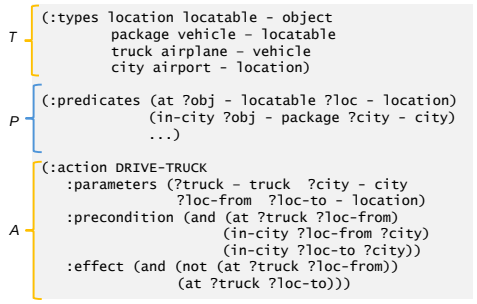
Figure 2a shows an excerpt from the Logistics domain that models delivering packages with trucks within cities and with planes between cities. The action "drive-truck" describing driving a truck between two locations is parametrized with variables "?truck", "?loc-from", "?loc-to" and "?city" whose instantiation with objects specifies the truck being driven, the start and the destination location and the city in which the locations are. The precondition states that the action can only be executed if the truck is at location ?loc-from and both locations are in city ?city. The effect makes the atom (at ?truck ?loc-from) false and (at ?truck ?loc-to) becomes true, i.e., it changes the state by moving the truck ?truck from ?loc-from to ?loc-to.

Variables can also have types restricting which objects can be used for their instantiation. For example, ?truck has the type "truck" which in our example has only one corresponding object "t0". There are different variants of the PDDL language with varying expressiveness. Here, we consider a variant of PDDL allowing typing of variables and restricted to conjunctive conditions with negations.

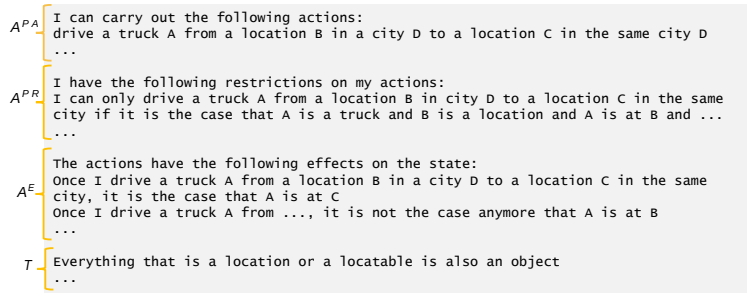
PDDL and LLMs. Valmeekam et al. [21] presented Planbench, a benchmark framework for assessing different aspects of reasoning capabilities of LLMs based on classical planning problems formulated in PDDL. Their assessment pipeline can take PDDL domain and problem files as input as well as natural language (NL) descriptions of the PDDL domain and problem files for assessing LLMs on NL problem formulations. For the NL inputs, they manually create a NL description of the domain file and handcraft NL translations for the individual PDDL actions, predicates and for object names. The individual translations are used to compose NL descriptions of problem files and plans.

In their assessment pipeline, Valmeekam et al. [21] prompt an LLM to generate a plan based on the NL descriptions of the domain, the initial and goal state and few-shot examples, i.e., example problems with their corresponding plans for in-context learning. The predicted NL action sequences get translated back into PDDL by

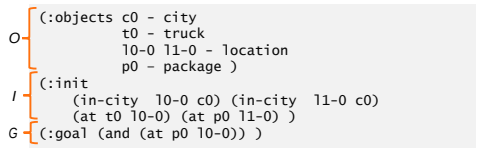
[1] We use the term "LLM action choice mechanism" here as a generic term encompassing both plan generation and action policies, and to emphasize that (in difference to symbolic planners) these mechanisms do not use any search, instead choosing actions directly.
[2] A similar observation was made by Silver et al. [19], but in a more limited setting considering PDDL prompts and a set-up comparable to our Basic LLM Planning configuration, which exhibits much worse performance than our strongest methods.
[3] https://github.com/minecraft-saar/autoplanbench
[4] https://www.icaps-conference.org/competitions/



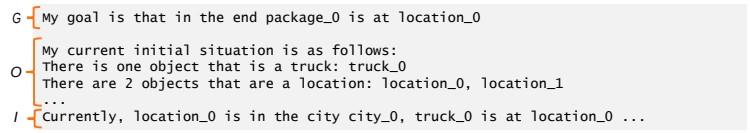
(a) PDDL domain definition with the type hierarchy ($\mathcal{T}$), predicates ($\mathcal{P}$), and the "drive-truck" action ($\mathcal{A}$).



(b) NL domain description consisting of the available actions with parameters ($\mathcal{A}^{\mathcal{PA}}$), their preconditions ($\mathcal{A}^{\mathcal{PR}}$) and effects ($\mathcal{A}^{\mathcal{E}}$) and the type hierarchy ($\mathcal{T}$).



(c) PDDL problem file stating the available objects with types ($\mathcal{O}$), the initial state ($\mathcal{I}$) and the goal condition ($\mathcal{G}$).



(d) NL problem description stating the goal ($\mathcal{G}$), the available objects ($\mathcal{O}$) and the initial state ($\mathcal{I}$).

Figure 2. Part of the Logistics PDDL domain file (2a), a problem file (2c) and the corresponding NL descriptions generated by AUTOPLANBENCH (2b, 2d).

a domain-dependent translator and are automatically evaluated by a plan validator. This process allows a systematic and objective evaluation of the performance of LLMs on different reasoning-related test cases. However, the approach includes several domain-dependent components. Extending the framework to new domains in addition to the three domains for which Valmeekam et al. [21] provide these components hence requires manual effort such as the creation of all NL descriptions for the target PDDL domain.

Silver et al. [19] assess the planning capabilities of LLMs when using prompts that do not contain any natural language and consist of the target problem definition and two few-shot examples in PDDL. They evaluate OpenAI's Codex LLM [2], an LLM specifically trained to generate code for NL inputs, and find that in some domains such as Gripper and Movie, LLMs can solve even large problems while they completely fail on more than half of the domains, including Blocksworld. Valmeekam et al. [22] also assess the LLM capabilities on PDDL but their prompts include a general task description in NL and the PDDL domain definition in addition to the target problem and examples. They find that for Blocksworld, using PDDL inputs leads to a drop in performance compared to NL whereas on Logistics they do not observe differences.

In addition to investigating the reasoning capabilities of LLMs on problems originally formulated in PDDL there has also been work investigating the usage of LLMs as an interface between problems in natural language and symbolic AI planning tools. Liu et al. [14] propose to use LLMs to translate NL descriptions of planning problems into PDDL. They then use a symbolic planner to find a plan in PDDL and translate the plans back into NL. Xie et al. [26] focus on letting LLMs translate goals specified in NL into goal definitions in PDDL.

3 Converting PDDL tasks into natural language

We contribute an automatic conversion of PDDL domains and problems into NL descriptions, to be used for prompting LLMs to choose actions. In contrast to previous works [e.g., 21] that created these NL descriptions manually for each individual planning domain, we introduce a framework, called AUTOPLANBENCH, that generates all

```
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 ... For actions, also include the type for each
parameter in the brackets if this information is available. ...
--------------------------------------------------------------------------
Original: (planet ?ob)
Output: {?ob} is a planet

Original: (undertree ?obj)
Output: {?obj} is under the tree

Original: (in ?obj ?loc)
Output: {?obj} is in {?loc}
...
--------------------------------------------------------------------------
Original: #PDDL PREDICATE#
Output:
```

Figure 3. Part of the prompt for converting PDDL predicates into NL consisting of the task description (top), few-shot examples (middle) and the target predicate (bottom).

mappings from PDDL objects, predicates and actions to their NL descriptions automatically. These PDDL-to-NL mappings are then joined together into a full prompt for an LLM, tasking the LLM to solve the described planning instance. To achieve this conversion from PDDL to NL automatically for any input PDDL task, we utilize an LLM (that we refer to as APB-LLM, for AUTOPLANBENCH-LLM), with few-shot prompting.

Figure 3 illustrates the input prompt received by the APB-LLM for translating PDDL predicates into NL. The prompt consists of three parts. The first part is a manually designed instruction explaining the PDDL-to-NL translation. The second part consists of several examples illustrating to the APB-LLM how to conduct the translation. These so-called few-shot examples are hand-crafted, but we use exactly the same examples independently of the target PDDL domain. We use seven examples of predicates of arity ranging from 0 to 2. The last part of the prompt consists of the actual predicate that we want to translate[5], i.e., we tell the APB-LLM which PDDL predicate we want to translate to NL (e.g., "Original: (at ?x ?y)") followed by "Output:" and we expect the APB-LLM to return its NL description (e.g., "{?x} is at {?y}"). Note that the NL descriptions are constructed as

[5] We use #PART# to mark parts of the shown prompts that are placeholders for actual content omitted for the presentation throughout the paper.

Table 1. Example PDDL-to-NL translation by the APB-LLM in the Logistics domain; predicates at the top; the "drive-truck" action at the bottom.

Input:	(truck ?truck)	(location ?location)
Output:	{?truck} is a truck	{?location} is a location
Input:	(at ?obj ?loc)	(in-city ?obj ?city)
Output:	{?obj} is at {?loc}	{?obj} is in the {?city}

Input:	action: drive-truck parameters: (?truck ?loc-from ?loc-to ?city) preconditions of drive-truck: ?truck is a truck and ?loc-from is a location and ?loc-to is a location and ?city is a city and ?truck is at ?loc-from and ?loc-from is in city ?city and ?loc-to is in city ?city effects of drive-truck: it becomes true that ?truck is at ?loc-to and it is not the case anymore that ?truck is at ?loc-from
Output:	drive truck {?truck} from location {?loc-from} in city {?city} to location {?loc-to} in the same city

templates with placeholders for actual objects (e.g., "{?obj}") which are replaced later when constructing the final NL task description. Table 1 (top) shows the NL descriptions of predicates generated in the Logistics domain.

NL descriptions of actions are generated analogously using a similar prompt but with different examples. In this case, each example consists of the name of the action, its parameters, and NL descriptions of the preconditions and effects that are constructed using the NL descriptions of the predicates generated by the APB-LLM as described above. The precondition is constructed by joining NL descriptions of its positive atoms by "and" and the conjoined negative preconditions are preceded by "it is not the case that". The NL descriptions of the positive (add) and negative (delete) effects are conjoined analogously. Moreover, we compile away parameter types using unary predicates [9]. We use the same four hand-crafted few-shot examples of actions independently of the target domain.

Table 1 (bottom) shows an example translation of the "drive-truck" action from the Logistics domain. It illustrates two interesting characteristics of our NL encodings. First, the order of the arguments in the generated NL encoding can deviate from the order of the parameters in the input PDDL domain. The order of PDDL parameters can be arbitrary and might not match a natural sounding or even syntactically correct order of arguments of the action verb in NL. We therefore include one few-shot example where the order deviates in the prompt for the APB-LLM to prevent the LLM from inferring that the order needs to be identical. Second, the generated NL description of "drive-truck" states the type of each parameter, i.e., it makes use of the information from preconditions to infer appropriate types. The complete conversion prompts can be found in Appendix A.

With the NL descriptions of predicates and actions in the form of templates, we can proceed with the generation of the domain and problem NL descriptions of the input PDDL task.

Figure 2b shows an excerpt from the NL *domain* description of the PDDL Logistics domain (Figure 2a). NL domain descriptions are designed to include the same information as the input PDDL. They start with the description of all possible actions ($A^{\mathcal{PA}}$), followed by their preconditions ($A^{\mathcal{PR}}$) and effects ($A^{\mathcal{E}}$). If the domain is typed, a verbalization of the type hierarchy is added also (T). Our template takes care of the statements introducing each part of the prompt (e.g. "I can carry out the following actions:") as well as of adequately composing the preconditions and effects into NL sentences (e.g. "Once I #ACTION# it is not the case anymore that #EFFECT#" for delete effects). The positive and negative preconditions are presented in two individual sentences. The same applies to the add and delete effects. Our composition method guarantees that all preconditions and effects are

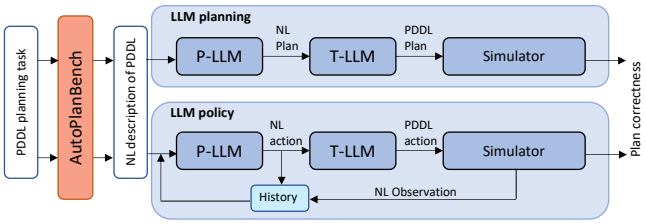


Figure 4. Overview of the set-up for the LLM plan generation and LLM action policy usage.

included in the final NL description. Moreover, we use a heuristic to add indefinite determiners to ensure that the domain encodings refer to objects in general instead of specific objects (e.g. "drive **a** truck A") and that the referring expressions allow to correctly infer which expressions refer to the same object.

The NL *problem* descriptions (see example in Figure 2c and 2d) consist of the specification of the goal condition (G), followed by available objects with their types (O), and the initial state (I).

The object names in PDDL problems can be any strings of characters and they often consist of single letters and numbers. For the NL description of the planning problems, more natural and semantically related object names are desirable. Our method generates new object names based on their types. If a domain is typed, we name each object after its (most specific) type and enumerate them, e.g. "t0" (Figure 2c) becomes "truck_0" (Figure 2d). Otherwise, we use the most general PDDL type "object" for all object names.

The NL descriptions of the initial state and goal are constructed using the NL descriptions of the corresponding predicates obtained by the APB-LLM. For example, the goal "(at p0 l0-0)" (Figure 2c) is converted into "package_0 is at location_0" and appended to "My goal is that in the end" (Figure 2d). If there is more than one goal fact, they are conjoined using "and". The description of the initial state is constructed analogously but starts with "Currently,".

4 LLM action choice mechanisms

Our automatic PDDL-to-NL translation can be used in concert with different LLM action-choice techniques. We distinguish LLM *planning* techniques (returning a whole action sequence at once) vs. LLM *policy* techniques (returning one action at a time); but they both consist of three core components, namely P-LLM, T-LLM and a simulator (see Figure 4). The NL description of the input PDDL task constructed using AUTOPLANBENCH described in Section 3 is passed to the LLM (denoted as P-LLM) responsible for the actual action choice (i.e., either a sequence of actions or a single action). The output of the P-LLM is in the form of NL. So, we pass its output to another LLM (denoted T-LLM) which translates the NL descriptions of actions (or action sequences) back to the PDDL format. Finally, we pass the PDDL actions returned by the T-LLM to the simulator responsible for validation and analysis of the system's action choice. We describe each component below.

Simulator. We implemented an environment simulator for the PDDL domain in AUTOPLANBENCH. The simulator makes use of the plan validator VAL[6] to determine whether an action is applicable in the current state, or to obtain the unsatisfied preconditions if it is not applicable. For LLM planning, the simulator only validates the output action sequence and determines whether it solves the task.

[6] https://github.com/KCL-Planning/VAL

For LLM policies, the simulator updates the world state after every action chosen by the LLM.

Instead of giving back the resulting world state in PDDL format, taking inspiration from prior work on non-PDDL forms of planning with LLMs [23, 18, 27, 13], our simulator produces a NL observation about actions' effects as feedback for the next LLM-policy step. If the action is applicable, the observation is a statement about the action being executed (e.g., "I drive truck truck_0 from ... to ..." for the NL action "Drive truck_0 from ... to ..."). This description is obtained by converting the PDDL action into its NL description using AUTOPLANBENCH. If the action is not applicable, our simulator states why this is the case, e.g., "I cannot drive truck truck_0 from location location_0 in city city_0 to ... because truck_0 is not at location_0." This type of feedback is constructed by converting the PDDL action and the unsatisfied preconditions into their NL descriptions using the template "I cannot #ACTION# because #UNSAT-PRE#" where each #PART# is, again, constructed using AUTOPLANBENCH.

Lastly, the simulator also determines whether the LLM's action choice reached the goal which is used as a termination condition for LLM policies.

T-LLM. The T-LLM translates the NL actions from the P-LLM output to valid actions in the PDDL format so that it can be further passed to the simulator. This translation is done by providing the T-LLM a prompt consisting of a statement providing information about the task (i.e., translating NL into PDDL) and the output format, followed by the NL descriptions of all actions of the domain obtained using AUTOPLANBENCH and the objects from the current problem. Lastly, up to five pairs of an NL action description from the domain and the corresponding PDDL action are included as few-shot examples. The domain-specific prompts are generated entirely automatically based on the generated PDDL-to-NL conversions and are identical for all action choice mechanisms (see Appendix D for details). This approach can be applied to any domain and is independent of the order of the verb and its arguments in the NL descriptions, hence allowing more flexibility than the domain-specific, rule-based translation approach used by Valmeekam et al. [21].

P-LLM. The P-LLM component implements the actual action choice mechanism. Figure 5 shows the structure of the prompts for the P-LLM in the LLM planning (left) and LLM policy (right) set-up. All prompts start with an instruction of the planning task (1), followed by the NL description of the goal (2) and the domain (3). Then a few-shot example from the same domain is included (4). It consists of the NL description of the goal of the example problem, the initial state and an example for the generation of an action or action sequence—this part is specific for each individual LLM action-choice method and we discuss it in detail below. The last part of the prompt consists of specific instructions for the LLM action-choice mechanisms (5), followed by the initial state of the target problem (6) (see complete prompts in Appendix D).

For LLM planning, the P-LLM predicts a complete sequence of NL actions, each in a separate line (see instructions (5) in Figure 5 (left)). The output of the P-LLM is translated into PDDL line-by-line by the T-LLM and then passed to the simulator that determines whether the generated action sequence is a valid plan for the task.

For LLM policies, the P-LLM generates a single action that is directly translated by the T-LLM and passed to the simulator which in turn produces a NL observation (or terminates the planning process in case the goal is satisfied). The generated NL observation is added to the history buffer (see Figure 4 (bottom)). In each step, the initial

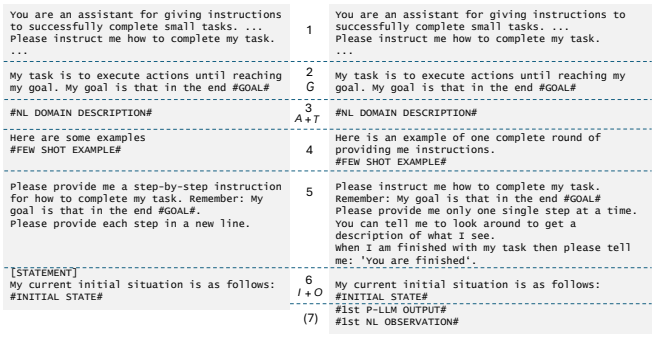


Figure 5. Structure of the prompts for the P-LLM in the LLM planning (left) set-up and in the policy set-up at the second prediction step (right).

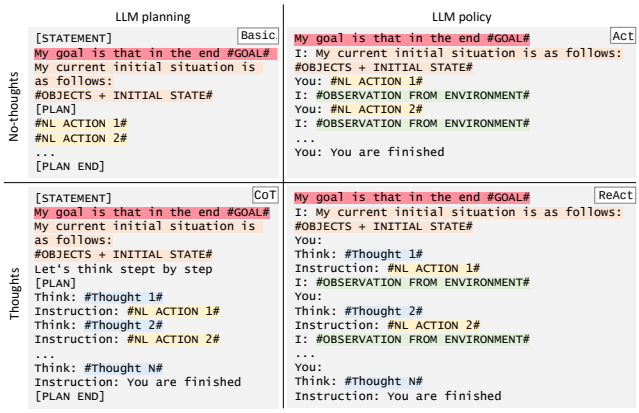


Figure 6. Structure of the few-shot examples for the four mechanisms.

prompt for the P-LLM is extended by all its previous outputs and observations from the history buffer (see part (7) in Figure 5 (right)). This approach provides the LLM access to the history of the action choice process as LLMs themselves do not include any memory, i.e., each call to an LLM is independent. Moreover, the process is not stopped when the P-LLM outputs an inapplicable action as the observations provide information that can be exploited by the LLM in subsequent steps. We also equip the P-LLM with an option to ask for the current state which is then provided by the simulator in the form of its NL description.

Action choice mechanisms. The action-choice instructions (part (5) in Figure 5) and few-shot examples (4) demonstrating the content and format expected from the P-LLM's output are specific to the individual LLM action-choice mechanisms. We focus on two LLM planning techniques (Basic and CoT) and two LLM policy techniques (Act and ReAct), described in what follows.

Basic. For the Basic configuration we prompt the P-LLM to generate a complete plan of NL actions. We follow Valmeekam et al. [22] and present each action of the example plan in a separate line and include a special tag to signal the beginning and end of the plan as shown in Figure 6 (top left).

CoT. Wei et al. [25] showed that prompting an LLM to generate a chain of thoughts, i.e., a sequence of explicit reasoning steps, improves the results on a range of reasoning tasks. The exact form and content of a thought are flexible and can include explicit reasoning over the current state, the next action or a goal fact to satisfy. For example, a thought for a Logistics problem could be "Now, package_0 is at truck_1 and truck_1 is at location_2 in city_4. Package_0 needs

to be moved to location_3 in city_4." The generation of thoughts by the P-LLM is elicited by adding thoughts between the actions in the few-shot example (see Figure 6, bottom left).

ReAct. ReAct combines CoT reasoning with information received from an environment and was proposed for interactive decision-making tasks. At each step, the P-LLM predicts a thought and an action and receives an observation from the simulator. The complete output of the P-LLM, i.e., including the thought, and the observation are added to the history buffer. As the length of the input increases with each step, the few-shot example does not demonstrate all individual steps but the last input (including the complete history) and output (see Figure 6, bottom right). In order to demonstrate which parts the P-LLM is supposed to generate, LLM outputs are preceded by the tag "You:" and parts provided by the simulator by "I:" in the few-shot examples.

Act. The Act mechanism works in the same way as ReAct but does not include reasoning thoughts (Figure 6, top right).

Note that all few-shot examples are generated automatically by converting one small example problem and its optimal plan into NL. For the LLM policy mechanisms, the simulator is used to generate the corresponding observations. For CoT and ReAct, reasoning thoughts are required. We use an LLM-based approach to obtain them: first the few-shot example in the ReAct structure is created (Figure 6, bottom right), but with placeholders instead of actual thoughts. For Logistics and Blocksworld we manually replace the placeholders by appropriate reasoning thoughts. We then use the one for Logistics in order to prompt an LLM to come up with appropriate thoughts to replace the placeholders for example problems from other domains. For the CoT few-shot examples the observations get removed afterwards (see Appendix E).

5 Experiments

The experimental evaluation aims at verifying that our automatically generated NL descriptions yield comparable performance to manual descriptions. Moreover, we demonstrate that our method allows us to effortlessly extend the evaluation of LLM action choice methods to a larger set of domains than used before. Lastly, we compare to several symbolic planning baselines to get a sense how well the LLM action choice methods work and scale with increasing size of tasks.

We use 12 classical planning domains. For each domain, we generate 21 solvable problems with optimal plan lengths between 3 and 20, select one of them as few-shot example and use the rest for evaluation. When available, we select an example problem with an optimal plan length of 4 or 5 and otherwise the one with the shortest plan (see supplementary material Appendix B for more details).

We run all four action-choice methods from Section 4: `Bas` (Basic), `CoT`, `ReAct` and `Act` and focus on GPT-4 [15] as the P-LLM and on few-shot prompting because previous work has shown these to be among the strongest currently available models for planning [e.g. 22]. We also use GPT-4 as the T-LLM and APB-LLM. The LLM policies `Act` and `ReAct` are not guaranteed to terminate and imposing a time limit is not an option here because we use GPT-4 via its API with high variability of response times. Therefore, we limit the LLM policies to 24 steps instead (which is at least twice the average of optimal length for all but one domain). Since we want to see how well our automatic translations work in comparison to the manual ones, we also evaluate all four LLM-based methods on the manual descriptions for the Blocksworld, Logistics and Depot domains created by Valmeekam et al. [22].

Table 2. Number of solved tasks. "Valm23" rows show results of the manual encodings by Valmeekam et al. [21] in the respective domains. We show in **bold** the best LLM-based method.

Domains	LLM with NL				PDDL		Symbolic Baselines			
	Bas	CoT	Act	ReAct	Bas	Act	rnd	BrFS	lmc	ff
Blocks. (20)	10	6	16	**19**	4	9	2.6	20	20	20
↳Valm23 (20)	9	11	18	18						
Logistics (20)	2	6	11	15	1	5	0.8	20	20	20
↳Valm23 (20)	3	6	**16**	**16**						
Depot (20)	1	3	4	**8**	0	0	0	20	20	20
↳Valm23 (20)	1	1	4	**8**						
Ferry (20)	3	10	12	**19**	4	10	1.0	20	20	20
Floortile (20)	0	0	0	0	0	0	0	18	20	20
Goldminer (20)	2	4	6	**8**	3	5	2.4	20	20	20
Grid (20)	3	4	14	**16**	1	5	1.2	20	20	20
Grippers (20)	8	15	15	**20**	4	9	0.4	20	20	20
Movie (20)	12	0	**20**	**20**	19	**20**	3.8	20	20	20
Rovers (20)	0	2	11	**12**	1	4	2.2	20	20	20
Satellite (20)	2	10	**18**	**18**	0	0	0	20	20	20
Visitall (20)	17	17	**20**	**20**	18	**20**	14.4	20	20	20
Σ (240)	60	77	147	**173**	55	87	28.8	238	240	240

Additionally, following the prior work of Silver et al. [19], we also conduct experiments with LLMs taking directly the PDDL input instead of its NL descriptions. We use only `Bas` and `Act` as it is not clear how to provide comparable thoughts for PDDL inputs. In this case, the T-LLM is skipped and the output of the P-LLM is passed directly to the simulator. Additionally, the observations are simplified to "Action was successfully executed." and "The action is not applicable in the current state." as the original observations rely on the PDDL-to-NL conversions. For a fairer comparison, we keep a slightly adapted task instruction as part of the prompts and only replace the NL domain, goal and initial state descriptions by their original PDDL input (see Appendix D for example prompts). This is closer to the way in which Valmeekam et al. [22] test PDDL inputs as they, in contrast to Silver et al. [19], include the PDDL domain description and also a short task description in natural language.

Lastly, we use several symbolic planning baselines. Breadth-first search (denoted as `BrFS`) is used to get a sense of hardness of tasks and how LLM-based methods compare to a basic uninformed search. We also compare to two basic state-of-the-art planners: optimal A* with the LM-Cut heuristic [10] (`lmc`), and satisficing greedy best-first search with FF heuristic [11] (`ff`). Lastly, to test whether LLM-based methods carry any information at all, we use a simple random search (`rnd`) limited by 24 steps (same as LLM policies) that, in every step, selects an applicable action uniformly at random. The results for `rnd` are averaged over five runs with different random seeds. These baselines were run on a cluster of Intel Xeon E5-2687W processors with 30 minutes and 8 GB time and memory limits, respectively.

Comparison of LLM action choice methods. Table 2 shows the number of solved tasks (coverage) per domain and overall. The comparison in Blocksworld, Logistics and Depot to manual descriptions (see "Valm23" rows) shows that using our automatic translations results in comparable performance. Exceptions are `CoT` in Blocksworld and `Act` in Logistics where manual descriptions work significantly better. The reason might be that the hand-crafted descriptions by Valmeekam et al. contain additional information that is not explicitly stated in PDDL (e.g., in their Blocksworld description, it is stated that a block is clear if no other block is on top of it) and that hence is also not stated in the automatically generated descriptions. However, when using the best-performing variant `ReAct` the results are on-par.

The results of `Bas` with the automatic translation over all domains support previous findings that a basic prompting technique does not work particularly well for plan generation [e.g., 21, 14]. Adding reasoning thoughts (`CoT`) improves performance substantially overall,

though the impact varies per domain and can also deteriorate performance. Using the LLM as an action policy instead of a plan generator in Act yields a major performance boost, dominating Bas and CoT consistently in every domain, with major coverage improvements in many of the domains. This shows that the use of LLMs, not for plan generation, but as *a part of plan generation* works much better—in the present case, the LLM being used for action choice only, with the computation of states being done symbolically and fed back into the LLM prompts. Adding reasoning thoughts to Act in ReAct yields another performance boost, consistently dominating coverage across all four LLM action choice mechanisms and achieving best LLM performance by far.

The comparison to LLM action choice with PDDL input (middle of Table 2) shows that the translation to NL is vastly superior overall, and is almost consistently beneficial. (A notable exception is Bas in Movie, where significantly more tasks are solved with PDDL input than with NL input.) There might be superior ways of using Act on PDDL inputs (e.g., using prompt engineering, different observations) that are not explored here. Nevertheless, our results strongly indicate that LLMs like GPT-4 tend to work better with NL inputs.

Our automatic PDDL-to-NL translation allows to assess the effect of PDDL vs. NL input types at a larger scale; and it allows to generate the better-performing NL inputs in domain-independent planning, providing the basis for deeper combination with symbolic planning methods in future work (we list some thoughts in Section 6).

Comparison to symbolic baselines. The comparison to the random baseline rnd clearly shows that LLM methods are able to extract at least some useful information from the task descriptions (with exception of Floortile where all LLM methods failed). This is further supported by the fact that the plans found by rnd were on average more than twice as long as the plans found by any of LLM methods.

As can be seen from the BrFS results, the evaluated tasks are fairly small, and yet LLM methods fail to solve them all. The performance of LLM-based methods significantly lag behind symbolic methods: lmc and ff solve all tasks, and even BrFS solves all tasks except for two in Floortile (the average runtime of lmc, ff and BrFS was 0.1, 0.7 and 1.7 seconds, respectively). This behaviour was already observed before—here, we provide a more comprehensive evaluation enabled by the automatic generation of NL descriptions. Nevertheless, we can also see some encouraging results with ReAct in Blocksworld, Ferry, Grippers, Movie and Visitall.

Scaling experiments. To see how far ReAct can scale in domains where it solved all (or almost all) tasks, we conduct more experiments with larger generated tasks. We focus on the Blocksworld, Ferry, Grippers, and Visitall domains. We leave out the Movie domain for which it is impossible to generate tasks with longer plans (scaling in this domain essentially only grows the initial state but the length of plans stays the same). The scaled data for the four considered domains is created as follows.

For each domain, we randomly generate a set of tasks, always varying only a single parameter: the number of blocks, cars, balls and locations for Blocksworld, Ferry, Grippers and Visitall, respectively (see Appendix B for more details). Then we run BrFS and lmc on each task with the same time and memory limit as in the previous experiments. For each domain, we identify the value N of the varied parameter (e.g., number of balls in Grippers) at which either BrFS or lmc is unable to solve the task. For the final dataset we select 20 problems per domain for which the scaled parameter values are around the identified threshold N. We use the same few-shot examples as in the first round of experiments. For the Grippers domain, we use only 11 tasks with the largest one including 20 balls. The reason is that the GPT-4 system has a limit on the number of tokens it can process (8 192 tokens) which was already reached by ReAct on the task with 20 balls.

Table 3. Number of solved tasks for selected domains scaled to larger instances.

Domains	NL	PDDL		Symbolic Baselines			
	ReAct	Bas	Act	rnd	BrFS	lmc	ff
Blocks. (20)	9	2	2	0	12	19	20
Ferry (20)	19	0	0	0	8	13	20
Grippers (11)	10	0	6	0	10	8	11
Visitall (20)	17	15	17	0	10	17	20
Σ (71)	55	17	25	0	40	57	71

Table 3 shows the coverage on the scaled benchmark set. We can, again, see that NL descriptions usually lead to much better performance than PDDL descriptions.

rnd cannot solve any benchmark instance at all here, confirming the vast superiority of ReAct as a much more informed action policy. BrFS is much better, but is also challenged by the size of these tasks. It is outperformed by ReAct overall, which is on-par in Grippers and much better in Ferry and Visitall. While BrFS is a very basic symbolic baseline, this provides additional evidence of ReAct's planning abilities. Indeed, remarkably, ReAct is on-par with the state-of-the-art optimal planner lmc in Visitall, and it even solves more tasks than lmc in Ferry and Grippers. This is remarkable given that ReAct, in contrast to lmc, does not perform any search. On the other hand, it should be noted that Ferry and Grippers are structurally simple domains, and that ReAct—in contrast to lmc—does not give any plan-optimality guarantee. The satisficing planner ff, which is comparable to ReAct in that regard, still has perfect coverage also on these scaled tasks, so the benchmarks are still not "hard enough" to be challenging for satisficing planning.

Overall, while LLM action choice at this point lags far behind symbolic planners, there are isolated islands of good performance, and our results do show promise for LLM planning abilities, in particular if used as part of a larger symbolic planning machinery (of which Act and ReAct are basic instances).

6 Conclusion

LLMs are rapidly gaining prominence in many sub-areas of AI, and the question if and how they can be applied in AI Planning is highly relevant. Following up on previous work in this direction, we show how to automate the conversion of PDDL into natural language prompts. Based on this, we contribute broad experiments, highlighting that the automatic conversion does not result in a performance loss relative to the previous hand-crafted prompts, and examining performance relative to representative symbolic methods. The results enhance our knowledge of LLM action choice performance, and demonstrate convincingly that LLMs do have *some* action-choice ability, outperforming random action selection and, in a few cases, even a state-of-the-art optimal planner. This performance is still far from the state of the art in symbolic (satisficing) planning, yet it is achieved without any search, pointing to the potential of more general uses of LLMs in planning.

The most direct question for future work, in our view, is how to combine LLMs with symbolic search methods. Our work lays the basis for that through the automatic translation of PDDL into natural language prompts, which as our results show boosts the LLM's planning ability. The space of possible combinations is vast. One could use the LLM to suggest preferred actions in search, one could search

around LLM-predicted plans or actions, one could apply plan repair to the LLM's suggestion (as suggested by [22] with LPG [3]), one could use LLM-generated plans as the basis for heuristic functions, etc. For further research on the question whether LLMs on their own (without search) can yield better planning performance, specialized training or neurosymbolic methods may be interesting to look at.

Acknowledgements

We thank Julia Wichlacz for fruitful discussions and valuable feedback. The work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 232722074 – SFB 1102. We gratefully acknowledge the stimulating research environment of the GRK 2853/1 "Neuroexplicit Models of Language, Vision, and Action", funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under project number 471607914.

References

[1] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng. Do as i can, not as i say: Grounding language in robotic affordances. In *6th Conference on Robot Learning (CoRL)*, 2022.

[2] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. Preprint arXiv:2107.03374, 2021.

[3] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs. 20:239–290, 2003.

[4] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language, 1998.

[5] T. Gupta and A. Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14953–14962, 2023.

[6] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[7] P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2019.

[8] M. Helmert. The fast downward planning system. *J. Artif. Intell. Res.*, 26:191–246, 2006.

[9] M. Helmert. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence*, 173(5-6):503–535, 2009.

[10] M. Helmert and C. Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS*. AAAI, 2009.

[11] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[12] T. Khot, D. Khashabi, K. Richardson, P. Clark, and A. Sabharwal. Text modular networks: Learning to decompose tasks in the language of existing models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1264–1279. Association for Computational Linguistics, 2021.

[13] B. Y. Lin, Y. Fu, K. Yang, F. Brahman, S. Huang, C. Bhagavatula, P. Ammanabrolu, Y. Choi, and X. Ren. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. In *Advances in Neural Information Processing Systems*, volume 36, pages 23813–23825. Curran Associates, Inc., 2023.

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

[15] OpenAI. Gpt-4 technical report. Preprint arXiv:2303.08774, 2023.

[16] A. Prasad, A. Koller, M. Hartmann, P. Clark, A. Sabharwal, M. Bansal, and T. Khot. Adapt: As-needed decomposition and planning with language models. Preprint arXiv:2311.05772, 2023.

[17] J. Seipp, Á. Torralba, and J. Hoffmann. PDDL generators. https://doi.org/10.5281/zenodo.6382173, 2022.

[18] M. Shridhar, X. Yuan, M.-A. Cote, Y. Bisk, A. Trischler, and M. Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

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

[20] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. P. Kaelbling, and M. Katz. Generalized planning in pddl domains with pretrained large language models. In *38th AAAI Conference on Artificial Intelligence (AAAI'24)*, 2024.

[21] K. Valmeekam, M. Marquez, A. Olmo, S. Sreedharan, and S. Kambhampati. 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*, 2023.

[22] K. Valmeekam, M. Marquez, S. Sreedharan, and S. Kambhampati. On the planning abilities of large language models - a critical investigation. In *Advances in Neural Information Processing Systems*, pages 75993–76005. Curran Associates, Inc., 2023.

[23] R. Wang, P. Jansen, M.-A. Côté, and P. Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11279–11298. Association for Computational Linguistics, 2022.

[24] Z. Wang, S. Cai, G. Chen, A. Liu, X. S. Ma, and Y. Liang. Describe, explain, plan and select: Interactive planning with llms enables open-world multi-task agents. In *Advances in Neural Information Processing Systems*, volume 36, pages 34153–34189. Curran Associates, Inc., 2023.

[25] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou. 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., 2022.

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

[27] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[28] X. Zhu, Y. Chen, H. Tian, C. Tao, W. Su, C. Yang, G. Huang, B. Li, L. Lu, X. Wang, Y. Qiao, Z. Zhang, and J. Dai. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. Preprint arXiv:2305.17144, 2023.

A Conversion prompts and examples

Figure 7 shows the template of the prompts used to derive the core natural language descriptions for the predicates and the actions based on which the overall domain and problem descriptions and the few-shot examples for the P-LLM are generated. For the APB-LLM i.e. for the conversion, we use the same few-shot examples for each input domain. The few-shot examples for both the predicate and the action conversion were created manually and are presented as part of the entire prompts shown in Figure 8 and 9. The last few-shot example

for the action "move" in Figure 9 was designed specifically in order to avoid that the LLM picks up the pattern that the order of parameters in the NL descriptions needs to match the order of parameters in PDDL.

Table 4 presents the NL descriptions generated by AUTOPLANBENCH for all PDDL predicates and actions of the Logistics domain as an example of the output of our conversion approach.

B PDDL domains and problems

For our experiments we select 12 PDDL domains from classical planning. We only select STRIPS domains with only "not" and "and" as operators and we remove the costs if the domain includes action costs. For the Blocksworld, Logistics and Depot domain we use the domain PDDL files and problems from the PlanBench repository[7]. For all other domains we take the domain files from the PDDL-Generators repository[8] [17] which offers a collection of planning domains and the problem generators for the domains. We use the available problem generators to generate the problems included in our dataset and select rather small values for the parameters that can be varied for the individual domains for creating the dataset for the main experiment. The task set-ups of the different domains are described in Table 5.

The Movie domain is the only domain to which we make changes beyond removing costs. In particular, as described in Table 5, we modify the domain such that the actions for buying snacks have additional preconditions that do not allow buying them in all states. Additionally, we introduce some variation to the planning task by generating not only problems where one object of each kind of snacks needs to be bought but also problems were less (but at least one) kinds of snacks must be bought.

Table 6 presents for each domain the optimal plan length, number of goal conditions, number of predicates in the initial state and the number of available objects averaged over the 21 problems used in the main experiments (including the few-shot example) and over the problems from the scaling experiments. Additionally, the last three columns present the number of unique actions, predicates and types for all of the domains. The optimal plans were generated by the Fast Downward planner [8] with the A* algorithm [6] and the LM-Cut heuristic [10].

Scaling experiments. For the scaling experiment, we focus on scaling only one specific type of object per domain. In the Blocksworld domain, the only parameter that can be scaled is the number of available blocks. For Grippers, we only vary the number of balls while fixing the number of robots to 1 and the number of rooms to 2, hence essentially restricting the problems to problems of the Gripper domain. For the Ferry domain, we fix the number of locations to 3 while varying the number of available cars. Finally, for the Visitall problems we vary the x and y dimension of the grid and restrict the problems to those where exactly the x*y locations need to be visited. For the Blocksworld, Gripper and Visitall domains always all objects of the varied type need to be part of a successful plan. For the Ferry domain, we select only problems where more than 80% of the cars need to be moved to achieve the goal.

[7] https://github.com/karthikv792/LLMs-Planning/tree/main/plan-bench
[8] https://github.com/AI-Planning/pddl-generators/tree/main

C Experimental set-up: Model parameters

All experiments with GPT-4 were conducted using the OpenAI API and the gpt-4-0613 model. The main experiments were conducted in the first half of November 2023 and the scaling experiments in the first half of March 2024. The experiments on PDDL inputs were conducted at the end of May and beginning of April 2024.

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 target initial states and different few-shot examples in the initial prompt and therefore the P-LLM never gets exactly the same input twice.

For all other parameters we keep the default values except for the maximum number of tokens:

APB-LLM: 50
P-LLM: no limitation
T-LLM: 256
LLM to generate thoughts for few-shot example: 300

D P-LLM and T-LLM prompts

Figure 10 and 11 present the prompt templates for the P-LLM in the LLM planning and LLM policy set-ups respectively. Figure 13 and Figure 14 show the complete prompts for running the Basic and ReAct action choice mechanisms on one of the problems from the Blocksworld domain. Additionally, Figure 12 presents the prompt template used for the Act LLM policy on PDDL inputs and 15 and 16 present examples of the complete prompts used for PDDL inputs for the Blocksworld domain.

The prompts for the T-LLM are automatically created by AUTOPLANBENCH from the template in Figure 17. For each domain, the pairs of PDDL and NL encodings of all actions of the domain are included in the prompt. Additionally, AUTOPLANBENCH creates few-shot examples for the translation from NL to PDDL by randomly sampling up to five distinct actions with different number of parameters if possible and sampling the required number of different objects from a list of example object names. The example object names were designed such that they are similar to object names from the actual planning problems but unlikely to overlap with them. The example objects are included in the list of available objects in the prompt which is extended by the actual objects of the specific problem instance. Except for the objects list, the prompt is identical for all problem instances of the same domain. In particular, there are no differences between the different action choice mechanisms with respect to the prompt for the T-LLM. Figure 18 shows an example of the prompt for problems in the Blocksworld domain.

E Automatic thought generation

The few-shot examples for the `ReAct` and `CoT` approach require the availability of appropriate reasoning thoughts for the steps in the example problem. In order to automate the few-shot example generation, we let GPT-4 generate these thoughts in advance. Figure 19 shows the structure of the prompt that is used for the thoughts generation: first, the instructions for the task are given, then a few-shot example for the task of generating thoughts is provided. The few-shot example is always the same example from the Logistics domain and consists of the NL domain description, the goal description, the

```
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.

#FEW-SHOT EXAMPLES#

Input: #TARGET PDDL PREDICATE / ACTION#
Output:
```

Figure 7. The prompt template for the APB-LLM used to generate the NL descriptions

```
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 8. The complete prompt for the APB-LLM, used to generate the NL descriptions for the PDDL predicates.

actions of the optimal plan, the observations from the simulator and placeholders for the thoughts. The actual thoughts are included below and were manually created by us.

Figure 20 shows the Logistics few-shot example in the same format in which it is included in the prompts of the P-LLM in the `ReAct` approach. The overall example is generated automatically by AUTOPLANBENCH except for the thoughts which we created manually. Figure 21 and 22 illustrate how the Logistics few-shot example is presented in the prompt for generating the thoughts for other domains (here the Blocksworld domain).

The problems selected as few-shot examples were shortened for the `CoT` and `ReAct` mechanisms in order to reduce the financial cost and the number of tokens. For each of the few-shot examples, only the last three steps of the plan for the original problem were kept. Additionally, we replaced the original initial state with the state obtained by executing all actions from the original plan that were removed for the example.

```
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: #ACTION#
parameters of #ACTION#: #PARAMETERS#
preconditions of #ACTION#: #NL PRECONDITIONS#
effects of #ACTION#: #NL EFFECTS#
Output:
```

Figure 9. The complete prompt for the APB-LLM, used to generate the NL descriptions for the PDDL actions.

Table 4. Natural language descriptions generated by AUTOPLANBENCH for the predicates and actions of the (untyped) Logistics domain.

PDDL	**NL description**
(OBJ ?obj)	{?obj} is an object
(TRUCK ?truck)	{?truck} is a truck
(LOCATION ?loc)	{?loc} is a location
(AIRPLANE ?airplane)	{?airplane} is an airplane
(AIRPORT ?airport)	{?airport} is an airport
(CITY ?city)	{?city} is a city
(at ?obj ?loc)	{?obj} is at {?loc}
(in ?obj1 ?obj2)	{?obj1} is in {?obj2}
(in-city ?obj ?city)	{?obj} is in the city {?city}
(load-truck ?obj ?truck ?loc)	load object {?obj} into truck {?truck} at location {?loc}
(load-airplane ?obj ?airplane ?loc)	load object {?obj} into airplane {?airplane} at location {?loc}
(unload-truck ?obj ?truck ?loc)	unload object {?obj} from truck ?truck at location {?loc}
(unload-airplane ?obj ?airplane ?loc)	unload object {?obj} from airplane {?airplane} at location {?loc}
(drive-truck ?truck ?loc-from ?loc-to ?city)	drive truck {?truck} from location {?loc-from} in city {?city} to location {?loc-to} in the same city
(fly-airplane ?airplane ?loc-from ?loc-to)	fly airplane {?airplane} from airport {?loc-from} to airport {?loc-to}

Table 5. Overview over the 12 different planning domains used in our experiments.

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.
Gripper	yes	The same as Grippers but with only one robot and two rooms where all balls are in the same room and need to be transported to the other room.
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	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 be 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 6. Overview over the characteristics of the problem instances per domain used for our main experiments (including the few-shot example) and of the problem instances from the scaling experiments. The right part of the table additionally shows some of the characteristics of the domains.

Domain	N	Coverage	Average				Number of		
		ReAct	Optimal plan length	#Goal facts	#Initial facts	#Objects	Actions	Predicates	Types
Blocksworld	21	19	7.05	2.10	6.90	4.00	4	7	1
scaled	20	9	22.40	6.05	13.60	9.80			
Depot	21	8	12.24	2.71	25.00	18.00	5	8	10
Ferry	21	19	8.95	4.62	20.76	7.67	3	9	1
scaled	20	19	44.95	16.55	44.10	19.55			
Floortile	21	0	14.10	5.76	38.57	13.90	7	12	4
Goldminer	21	8	10.62	1.00	33.52	8.14	7	14	2
Grid	21	16	8.14	1.33	54.71	13.29	5	14	1
Grippers	21	20	9.24	4.57	9.71	13.43	3	6	4
scaled	11	10	44.45	15.00	18.00	20.00			
Logistics	21	15	8.43	1.95	22.76	11.38	6	11	1
Movie	21	20	6.48	5.48	11.10	10.10	9	15	1
Rovers	21	12	10.43	3.14	35.05	13.19	9	27	8
Satellite	21	18	7.81	2.76	12.43	10.67	5	10	5
Visitall	21	20	4.57	4.33	14.00	6.00	1	5	2
scaled	20	17	30.70	31.45	99.20	31.45			

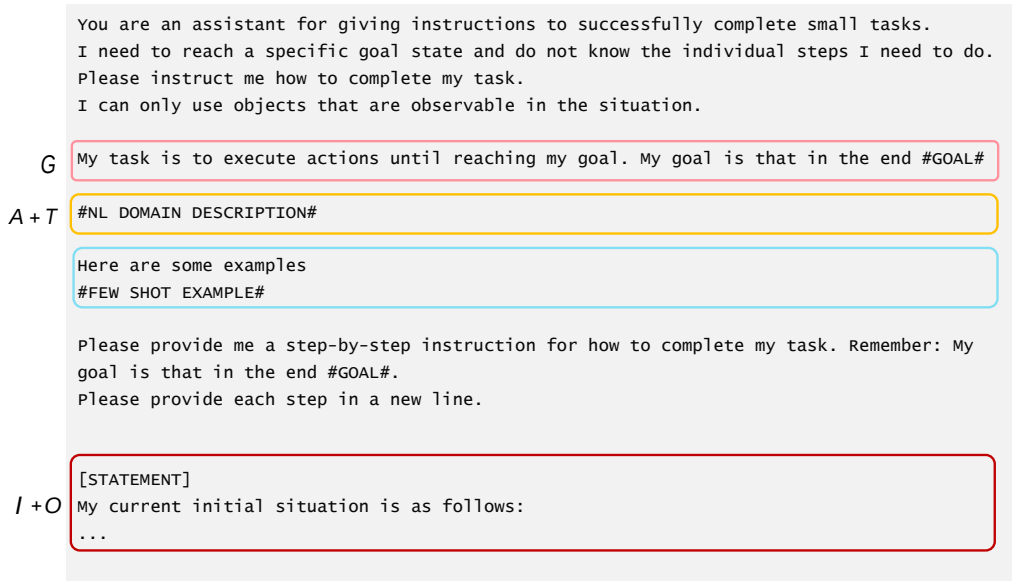


Figure 10. Prompting template used for the LLM planning mechanisms. #GOAL# corresponds to the problem-specific goal description, #NL DOMAIN DESCRIPTION# to the domain-specific description and the #FEW SHOT EXAMPLE# is the approach-specific planning example. At the bottom, the beginning of the target problem description is shown.

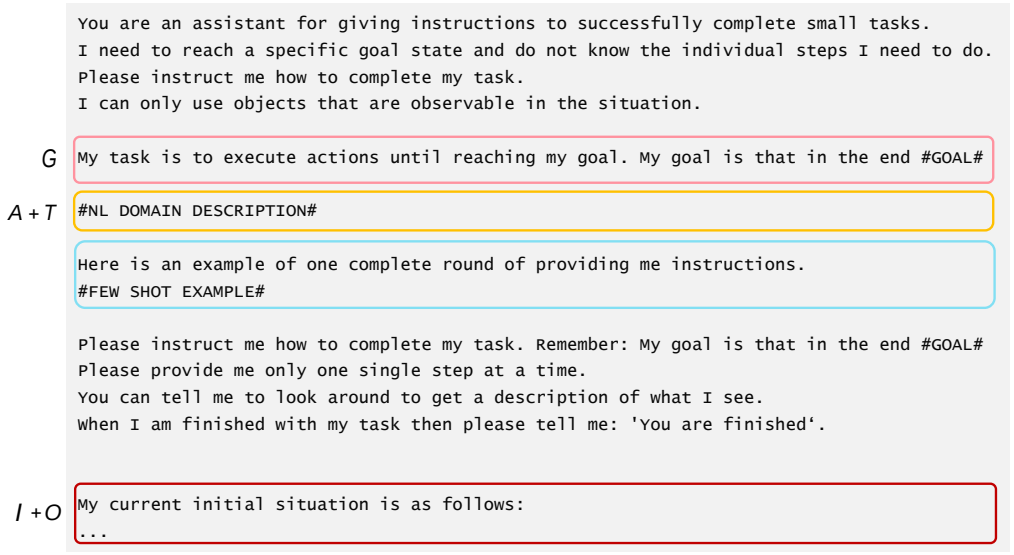


Figure 11. Prompting template used for the LLM policy mechanisms. #GOAL# corresponds to the problem-specific goal description, #NL DOMAIN DESCRIPTION# to the domain-specific description and the #FEW SHOT EXAMPLE# is the approach-specific planning example. At the bottom, the beginning of the target problem description is shown.

```
You are an assistant for providing a plan in the pddl language 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.
```

G

```
My task is to execute actions until reaching the goal. The goal is:
#PDDL GOAL#
```

A + T

```
#PDDL DOMAIN DESCRIPTION#
```

```
Here is an example of one complete round of providing a plan:
#FEW SHOT EXAMPLE#
```

```
Please instruct me how to complete my task. Remember the goal is:
#PDDL GOAL#

Please provide only one single step at a time.
```

I +O

```
(:objects ...)
(:init ...)
```

Figure 12. Prompting template used for running the Act LLM policy mechanism on PDDL inputs.

```
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.
```

G

```
My task is to execute actions until reaching my goal. My goal is that in the end object_0 is on
object_2
```

A

```
I can carry out the following actions:
pick up an object A from the table
put down an object A
stack an object A on top of an object B
unstack an object A from an object B

I have the following restrictions on my actions:
I can only pick up an object A from the table if it is the case that A is clear and the hand is
empty and A is on the table
I can only put down an object A if it is the case that A is being held
I can only stack an object A on top of an object B if it is the case that A is being held and B
is clear
I can only unstack an object A from an object B if it is the case that A is clear and A is on B
and the hand is empty

The actions have the following effects on the state:
Once I pick up an object A from the table, it is the case that A is being held
Once I pick up an object A from the table, it is not the case anymore that A is clear and it is
not the case anymore that the hand is empty and it is not the case anymore that A is on the
table
Once I put down an object A, it is the case that A is on the table and A is clear and the hand
is empty
Once I put down an object A, it is not the case anymore that A is being held
Once I stack an object A on top of an object B, it is the case that A is clear and A is on B and
the hand is empty
Once I stack an object A on top of an object B, it is not the case anymore that A is being held
and it is not the case anymore that B is clear
Once I unstack an object A from an object B, it is the case that A is being held and B is clear
Once I unstack an object A from an object B, it is not the case anymore that A is clear and it
is not the case anymore that A is on B and it is not the case anymore that the hand is empty
```

```
Here are some examples
[STATEMENT]
My goal is that in the end object_2 is on object_1
My current initial situation is as follows:
There are 4 objects: object_0, object_1, object_2, object_3
Currently, object_0 is clear, object_0 is on the table, object_1 is clear, object_1 is on
object_2, object_2 is on the table, object_3 is clear, object_3 is on the table, the hand is
empty
[PLAN]
unstack object object_1 from object object_2
Now, put down object object_1
Next, pick up object object_2 from the table
Next, stack object object_2 on top of object object_1
[PLAN END]
```

```
Please provide me a step-by-step instruction for how to complete my task. Remember: My goal is
that in the end object_0 is on object_2.
Please provide each step in a new line.
```

I+O

```
[STATEMENT]
My current situation is as follows:
There are 4 objects: object_0, object_1, object_2, object_3
Currently, object_0 is clear, object_0 is on object_3, object_1 is on object_2, object_2 is on
the table, object_3 is on object_1, the hand is empty
[PLAN]
```

Figure 13. The complete input prompt for the `Basic` LLM planning mechanism for one of the Blocksworld problems. The prompt consists of the overall instruction for the LLM, the goal description of the problem, the Blocksworld domain descriptions, one few-shot example, the final planning-specific instructions and the initial state of the target problem.

```
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.
```

G

```
My task is to execute actions until reaching my goal. My goal is that in the end object_0 is on
object_2
```

A

```
I can carry out the following actions:
pick up an object A from the table
put down an object A
stack an object A on top of an object B
unstack an object A from an object B

I have the following restrictions on my actions:
I can only pick up an object A from the table if it is the case that A is clear and the hand is
empty and A is on the table
I can only put down an object A if it is the case that A is being held
I can only stack an object A on top of an object B if it is the case that A is being held and B
is clear
I can only unstack an object A from an object B if it is the case that A is clear and A is on B
and the hand is empty

The actions have the following effects on the state:
Once I pick up an object A from the table, it is the case that A is being held
Once I pick up an object A from the table, it is not the case anymore that A is clear and it is
not the case anymore that the hand is empty and it is not the case anymore that A is on the
table
Once I put down an object A, it is the case that A is on the table and A is clear and the hand
is empty
Once I put down an object A, it is not the case anymore that A is being held
Once I stack an object A on top of an object B, it is the case that A is clear and A is on B and
the hand is empty
Once I stack an object A on top of an object B, it is not the case anymore that A is being held
and it is not the case anymore that B is clear
Once I unstack an object A from an object B, it is the case that A is being held and B is clear
Once I unstack an object A from an object B, it is not the case anymore that A is clear and it
is not the case anymore that A is on B and it is not the case anymore that the hand is empty
```

```
Here is an example of one complete round of providing me instructions.
 My goal is that in the end object_1 is on the table, object_2 is on object_1, object_3 is on
the table.
I: My current initial situation is as follows:
There are 3 objects: object_1, object_2, object_3
Currently, object_1 is on the table, object_2 is on object_1, object_3 is on object_2, object_3
is clear, the hand is empty
You:
     Think: Currently, object object_1 is on the table, object object_2
          is on object object_1, object object_3 is on object object_2, object object_3 is
          clear, the hand is empty. The object object_3 needs to be moved on the table.
     Instruction: unstack object object_3 from object object_2
I: I unstack object object_3 from object object_2
You:
     Think: Now, object object_3 is being held and object object_1 is
          on the table, object object_2 is on object object_1, object_2 is clear.
     Instruction: put down object object_3
I: I put down object object_3
You:
     Think: object_1 is on the table, object_3 is on the table and
          object_2 is on object_1.
     Instruction: You are finished.
I: Great!
```

```
Please instruct me how to complete my task. Remember: My goal is that in the end object_0 is on
object_2.
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‘.
```

I+O

```
My current situation is as follows:
There are 4 objects: object_0, object_1, object_2, object_3
Currently, object_0 is clear, object_0 is on object_3, object_1 is on object_2, object_2 is on
the table, object_3 is on object_1, the hand is empty
```

Figure 14. The complete input prompt for the ReAct LLM planning mechanism for one of the Blocksworld problems. The prompt consists of the overall instruction for the LLM, the goal description of the problem, the Blocksworld domain descriptions, one few-shot example, the final planning-specific instructions and the initial state of the target problem.

```
You are an assistant for providing a plan in the pddl language 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.
```

G

```
My task is to execute actions until reaching the goal. The goal is:
(:goal
(and
(on object_2 object_0)))
```

A

```
Here is the definition of the domain in PDDL including the possible actions I can execute:
(define (domain blocksworld-4ops)
  (:requirements :strips)
(:predicates (clear ?x)
             (ontable ?x)
             (handempty)
             (holding ?x)
             (on ?x ?y))

(:action pick-up
  :parameters (?ob)
  :precondition (and (clear ?ob) (ontable ?ob) (handempty))
  :effect (and (holding ?ob) (not (clear ?ob)) (not (ontable ?ob))
               (not (handempty))))

(:action put-down
  :parameters  (?ob)
  :precondition (holding ?ob)
  :effect (and (clear ?ob) (handempty) (ontable ?ob)
               (not (holding ?ob))))

(:action stack
  :parameters  (?ob ?underob)
  :precondition (and (clear ?underob) (holding ?ob))
  :effect (and (handempty) (clear ?ob) (on ?ob ?underob)
               (not (clear ?underob)) (not (holding ?ob))))

(:action unstack
  :parameters  (?ob ?underob)
  :precondition (and (on ?ob ?underob) (clear ?ob) (handempty))
  :effect (and (holding ?ob) (clear ?underob)
               (not (on ?ob ?underob)) (not (clear ?ob)) (not (handempty)))))
```

```
Here are some examples
[STATEMENT]
 (:goal
(and
(on object_2 object_1)))

(:objects  object_0 object_1 object_2 object_3 )
(:init
(handempty)
(ontable object_0)
(on object_1 object_2)
(ontable object_2)
(ontable object_3)
(clear object_0)
(clear object_1)
(clear object_3))
[PLAN]
(unstack object_1 object_2)
(put-down object_1)
(pick-up object_2)
(stack object_2 object_1)
[PLAN END]
```

```
Please provide me a step-by-step instruction for how to complete my task. Remember the goal is:
(:goal
(and
(on object_2 object_0)))

Please provide each step in a new line.
```

I +O

```
(:objects ...)
(:init ...)
```

Figure 15. The complete input prompt for the `Basic` LLM planning approach for one of the Blocksworld problems using the PDDL descriptions and consisting of the overall instruction for the LLM, the goal of the problem, the Blocksworld domain descriptions, one few-shot example and final task instructions.

```
You are an assistant for providing a plan in the pddl language 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.
```

G

```
My task is to execute actions until reaching the goal. The goal is:
(:goal
(and
(on object_2 object_0)))
```

A

```
Here is the definition of the domain in PDDL including the possible actions I can execute:
(define (domain blocksworld-4ops)
  (:requirements :strips)
(:predicates (clear ?x)
             (ontable ?x)
             (handempty)
             (holding ?x)
             (on ?x ?y))

(:action pick-up
  :parameters (?ob)
  :precondition (and (clear ?ob) (ontable ?ob) (handempty))
  :effect (and (holding ?ob) (not (clear ?ob)) (not (ontable ?ob))
               (not (handempty))))

(:action put-down
  :parameters  (?ob)
  :precondition (holding ?ob)
  :effect (and (clear ?ob) (handempty) (ontable ?ob)
               (not (holding ?ob))))

(:action stack
  :parameters  (?ob ?underob)
  :precondition (and (clear ?underob) (holding ?ob))
  :effect (and (handempty) (clear ?ob) (on ?ob ?underob)
               (not (clear ?underob)) (not (holding ?ob))))

(:action unstack
  :parameters  (?ob ?underob)
  :precondition (and (on ?ob ?underob) (clear ?ob) (handempty))
  :effect (and (holding ?ob) (clear ?underob)
               (not (on ?ob ?underob)) (not (clear ?ob)) (not (handempty)))))
```

```
Here is an example of one complete round of providing a plan:
 The goal is:
(:goal
(and
(on object_2 object_1)))

I: (:objects  object_0 object_1 object_2 object_3 )
(:init
(handempty)
(ontable object_0)
(on object_1 object_2)
(ontable object_2)
(ontable object_3)
(clear object_0)
(clear object_1)
(clear object_3))
You: (unstack object_1 object_2)
I: Action was successfully executed.
You: (put-down object_1)
I: Action was successfully executed.
You: (pick-up object_2)
I: Action was successfully executed.
You: (stack object_2 object_1)
I: Action was successfully executed.
```

```
Please instruct me how to complete my task. Remember the goal is:
(:goal
(and
(on object_2 object_0)))

Please provide only one single step at a time.
```

I +O

```
(:objects ...)
(:init ...)
```

Figure 16. The complete input prompt for the `Act` LLM planning approach for one of the Blocksworld problems using the PDDL descriptions and consisting of the overall instruction for the LLM, the goal of the problem, the Blocksworld domain descriptions, one few-shot example and final task instructions.

```
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:
#DOMAIN-SPECIFIC ACTION DEFINITIONS IN PDDL AND NL#

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:
#EXAMPLE OBJECTS#
...
#OBJECTS FROM PROBLEM INSTANCE#
...

Here are some examples for your task:
#FEW-SHOT EXAMPLES#

Input: #ACTION PREDICTED BY P-LLM#
Output:
```

Figure 17. The prompt template for the T-LLM to translate the predicted natural language actions back into PDDL. The prompt consists of the general task instruction at the beginning followed by the domain-specific pairs of PDDL and NL action descriptions. All available objects are included in the prompt as well as some example objects that are used in the few-shot examples provided.

```
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


(stack ?ob ?underob)
description: stack object {?ob} on top of object {?underob}
...

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)

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 18. The prompt for the T-LLM to translate the predicted natural language actions from the Blocksworld domain back into PDDL.

```
You are a brilliant and helpful assistant to provide reasoning thoughts for
an interaction where one agent instructs another agent to execute a task.

You will be shown the actions that can be carried out, their preconditions
and their effects.

Additionally, you will see one interaction between the instruction giver
and the instruction follower.

Your task is to come up with an appropriate and good reasoning thought with
which [TODO: ADD REASONING THOUGHT] should be replaced.

#LOGISTICS NL DOMAIN DESCRIPTION#

#DESCRIPTION OF LOGISTICS REACT EXAMPLE PROBLEM WITH PLACEHOLDERS FOR
REASONING THOUGHTS#

#MANUALLY CREATED REASONING THOUGHTS FOR LOGISTICS EXAMPLE#

#TARGET NL DOMAIN DESCRIPTION#

#DESCRIPTION OF REACT EXAMPLE PROBLEM FROM TARGET DOMAIN WITH PLACEHOLDERS
FOR REASONING THOUGHTS#

The following are good reasoning thoughts for the TODOs:
```

Figure 19. Prompt template for generating the thoughts for the ReAct and CoT few-shot examples using an LLM. The LLM is provided a general task instruction followed by the manually created Logistics few-shot examples presented in Figure 20 and the domain description and gold plan trajectory of the target problem.

```
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 20. The manually created ReAct few-shot example for the Logistics domain encoded by AUTOPLANBENCH.

```
You are a brilliant and helpful assistant to provide reasoning thoughts for an interaction
where one agent instructs another agent to execute a task.

You will be shown the actions that can be carried out, their preconditions and their effects.

Additionally, you will see one interaction between the instruction giver and the instruction
follower.

Your task is to come up with an appropriate and good reasoning thought with which [TODO: ADD
REASONING THOUGHT] should be replaced.

I can carry out the following actions:
load an object A into a truck B at a location C
load an object A into an airplane B at a location C
unload an object A from a truck B at a location C
unload an object A from an airplane B at a location C
drive a truck A from a location B in a city D to a location C in the same city
fly an airplane A from an airport B to an airport C

I have the following restrictions on my actions:
I can only load an object A into a truck B at a location C if it is the case that B is at C
and A is at C and B is a truck and A is an object and C is a location
I can only load an object A into an airplane B at a location C if it is the case that A is at
C and B is at C and A is an object and B is an airplane and C is a location
I can only unload an object A from a truck B at a location C if it is the case that A is in B
and B is at C and B is a truck and A is an object and C is a location
I can only unload an object A from an airplane B at a location C if it is the case that B is
at C and A is an object and B is an airplane and A is in B and C is a location
I can only drive a truck A from a location B in a city D to a location C in the same city if
it is the case that B is a location and A is a truck and C is in the city D and A is at B and
C is a location and D is a city and B is in the city D
I can only fly an airplane A from an airport B to an airport C if it is the case that A is at
B and B is an airport and C is an airport and A is an airplane

The actions have the following effects on the state:
Once I load an object A into a truck B at a location C, it is the case that A is in B
Once I load an object A into a truck B at a location C, it is not the case anymore that A is
at C
Once I load an object A into an airplane B at a location C, it is the case that A is in B
Once I load an object A into an airplane B at a location C, it is not the case anymore that A
is at C
Once I unload an object A from a truck B at a location C, it is the case that A is at C
Once I unload an object A from a truck B at a location C, it is not the case anymore that A is
in B
Once I unload an object A from an airplane B at a location C, it is the case that A is at C
Once I unload an object A from an airplane B at a location C, it is not the case anymore that
A is in B
Once I drive a truck A from a location B in a city D to a location C in the same city, it is
the case that A is at C
Once I drive a truck A from a location B in a city D to a location C in the same city, it is
not the case anymore that A is at B
Once I fly an airplane A from an airport B to an airport C, it is the case that A is at C
Once I fly an airplane A from an airport B to an airport C, it is not the case anymore that A
is at B
```

Figure 21. Example of the prompt for generating the thoughts for the ReAct and CoT few-shot examples based on an example from the Logistics domain. The domain descriptions for the Logistics domain and for the target domain are those generated by AUTOPLANBENCH, the Logistics interaction example with the placeholders is generated automatically as well and the example reasoning thoughts were created manually.

```
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: [TODO: ADD REASONING THOUGHT]
     Instruction: drive truck object_1 from location object_2 in city object_4 to location
object_3 in the same city
I: I drive truck object_1 from location object_2 in city object_4 to location object_3 in the
same city
You:
     Think: [TODO: ADD REASONING THOUGHT]
     Instruction: unload object object_0 from truck object_1 at location object_3
I: I unload object object_0 from truck object_1 at location object_3
You:
     Think: [TODO: ADD REASONING THOUGHT]
     Instruction: You are finished.
I: Great!

The following are good reasoning thoughts for the TODOs:
0. 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.
1. 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.
2. Think: Now, the object object_0 is at location object_3
```

```
I can carry out the following actions:
pick up an object A from the table
...
My goal is that ...
I: ...
...

The following are good reasoning thoughts for the TODOs:
```

Figure 22. Continuation of the prompt in Figure 21