Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)

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

Katharina Stein[1], **Daniel Fišer**[3], **Jörg Hoffmann**[1,2], **Alexander Koller**[1]

[1]Saarland Informatics Campus, Saarland University, Saarbrücken Germany
[2]German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
[3]Aalborg University, Denmark
{kstein,koller}@lst.uni-saarland.de, danfis@danfis.cz, hoffmann@cs.uni-saarland.de

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 PDDL planning, based on manually converting three PDDL domains into natural language (NL) prompts. Here we automate this conversion step, showing how to leverage an LLM to automatically generate NL prompts from PDDL input. Our automatically generated NL prompts result in similar LLM-planning performance as the previous manually generated ones. Beyond this, the automation enables us to run much larger experiments, providing for the first time a broad evaluation of LLM planning performance in PDDL. Our NL prompts yield better performance than PDDL prompts and simple template-based NL prompts. Compared to symbolic planners, LLM planning lags far behind; but in some domains, our best LLM configuration scales up further than A* using LM-cut.

Code and datasets — https://github.com/minecraft-saar/autoplanbench

Extended version — https://arxiv.org/abs/2311.09830

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 (Ahn et al. 2022), text-based games (Yao et al. 2023) or Minecraft (Wang et al. 2023; Zhu et al. 2023), but also less structured tasks such as question answering (e.g., Wei et al. 2022; Khot et al. 2021), visual programming (Gupta and Kembhavi 2023), and the orchestration of API calls (Prasad et al. 2024).

Here we address PDDL planning. 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) (Silver et al. 2022; Valmeekam et al. 2023a,b); recent work explored the generation of program code for generalized planning (Silver et al. 2024). Here, we follow up on the prominent work line by Valmeekam et al. (2023a,b), who investigated the ability of LLMs to produce plans for PDDL planning tasks. Valmeekam et al. experiment with three wide-spread benchmark domains, Blocksworld, Depots and Logistics, for each of which they manually engineer natural language (NL) descriptions of the actions and predicates. They ask the LLM to produce a plan, thus serving as a form of 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.

We extend Valmeekam et al.'s work by automating the conversion of PDDL into NL prompts for LLM plan generation (and, more generally, action choice mechanisms, see below). We thus turn the use of LLMs with NL prompts into domain-independent planning machinery that does not rely on any input other than the PDDL itself. A key challenge here is to ensure that the (syntactic and semantic) relationship between an action and its arguments is captured correctly, and that PDDL object types are made explicit in the action descriptions in an adequate way.

We address this by leveraging LLMs themselves to support the conversion from PDDL to NL. We first use GPT-4o (OpenAI 2024) to convert PDDL predicates into NL snippets based on a few generic conversion examples. We then generate NL snippets for PDDL action schemas, by first converting the preconditions and effects based on a simple composition of the previously generated predicate snippets, and then doing the final conversion step with an LLM, based again on a few generic examples. Lastly, we compose the final NL description of the PDDL task for the LLM prompt from predicate and action snippets together with NL descriptions of initial state and goal. Our conversion approach ensures that, structurally, all predicate and actions schemas are preserved correctly in the NL descriptions. Experimenting with the same benchmarks used by Valmeekam et al., we find that our automatically generated prompts result in similar performance as the previous manually generated ones.

Beyond this, we provide the first broad evaluation of LLM performance in PDDL planning, comparing our automatically-generated NL prompts to PDDL prompts and simple template-based prompts, comparing four variants of LLM action choice mechanisms[1], and comparing LLM

[1]We use "LLM action choice mechanism" here as a generic

planning to several symbolic planning baselines, on a broad range of benchmark domains.

Specifically, we experiment with:

- **Basic LLM Planning:** Domain and problem description provided once, LLM generates a plan. Same as Valmeekam et al. (2023a,b).
- **CoT LLM Planning:** The LLM also generates a plan, but the prompt is enriched by Chain-of-Thought (CoT) prompting (Wei et al. 2022), asking the LLM to generate "thoughts" between the predicted actions in the plan. Valmeekam et al. (2023b) experimented with a simple version of this, producing the states before and after 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. (2023) in non-PDDL-planning contexts.)
- **ReAct:** Inspired by Yao et al. (2023); uses CoT within Act, generating intermediate thoughts between actions.

Our primary experiment is on 13 domains for which we generate small instances, targeting the relevant scaling range. For a subset of configurations, we also run experiments on all IPC domains, with their original instances, to the extent feasible (monetary cost for LLM calls is the main issue here). We provide comparisons to (1) LLM action choice based on PDDL prompts (Valmeekam et al. 2023b; Silver et al. 2022) rather than NL encodings thereof; (2) a simple template-based method to convert PDDL into NL prompts; (3) random action selection as a sanity test; (4) blind breadth-first search as a trivial symbolic baseline; as well as (5) strong optimal and satisficing planner baselines (Helmert and Domshlak 2009; Hoffmann and Nebel 2001).

Compared to the alternative automatic prompting methods (1) and (2), our NL prompts often yield better performance, in particular as they allow to incorporate Chain-of-Thought (in CoT and ReAct). All four LLM action choice mechanisms soundly beat (3) random action choice, showing that the LLM does carry *some* information about general PDDL planning (not self-evident given the paucity of internet text about most planning benchmark domains).[2] The comparisons (4) and (5) to symbolic planners are less favorable. 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 7 domains and the optimal planner in 6 domains. 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.

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. (2022), but in a more limited setting considering PDDL prompts and a setup comparable to our Basic LLM Planning configuration, which exhibits much worse performance than our strongest methods.

2 Background

PDDL (Ghallab et al. 1998) has been established in the context of the International Planning Competitions[3]. A planning 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. 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 1a shows an excerpt from the Logistics benchmark 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 PDDL subset allowing variable typing and restricted to conjunctive conditions with negation.

PDDL and LLMs. Valmeekam et al. (2023a) introduced 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. (2023a) 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 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

[3]https://www.icaps-conference.org/competitions/

```
(:types location locatable - object
        package vehicle - locatable
        truck airplane - vehicle
        city airport - location)
```
($\mathcal{T}$)

```
(:predicates (at ?obj - locatable ?loc - location)
             (in-city ?obj - package ?city - city)
             ...)
```
($\mathcal{P}$)

```
(:action DRIVE-TRUCK
   :parameters (?truck - truck  ?city - city
                ?loc-from  ?loc-to - location)
   :precondition (and (at ?truck ?loc-from)
                      (in-city ?loc-from ?city)
                      (in-city ?loc-to ?city))
   :effect (and (not (at ?truck ?loc-from))
                (at ?truck ?loc-to)))
```
($\mathcal{A}$)

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

```
I can carry out the following actions:
drive a truck A from a location B in a city D to a location C in the same city D
...
```
($\mathcal{A}^{\mathcal{PA}}$)

```
I have the following restrictions on my actions:
I can only drive a truck A from a location B in city D to a location C in the same
city if it is the case that A is a truck and B is a location and A is at B and ...
...
```
($\mathcal{A}^{\mathcal{PR}}$)

```
The actions have the following effects on the state:
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 ..., it is not the case anymore that A is at B
...
```
($\mathcal{A}^{\mathcal{E}}$)

```
Everything that is a location or a locatable is also an object
...
```
($\mathcal{T}$)

(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 ($\mathcal{T}$).

```
(:objects c0 - city
          t0 - truck
          l0-0 l1-0 - location
          p0 - package )
(:init
    (in-city  l0-0 c0) (in-city  l1-0 c0)
    (at t0 l0-0) (at p0 l1-0) )
(:goal (and (at p0 l0-0)) )
```
($\mathcal{O}$, $\mathcal{I}$, $\mathcal{G}$)

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

```
My goal is that in the end package_0 is at location_0
```
($\mathcal{G}$)

```
My current initial situation is as follows:
There is one object that is a truck: truck_0
There are 2 objects that are a location: location_0, location_1
...
```
($\mathcal{O}$)

```
Currently, location_0 is in the city city_0, truck_0 is at location_0 ...
```
($\mathcal{I}$)

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

Figure 1: Part of the Logistics PDDL domain and problem file and the NL descriptions generated by PDDL2NL.

reasoning-related test cases. However, the approach includes several manually provided per-domain components.

Silver et al. (2022) 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 (Chen et al. 2021), 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 evaluated domains, including Blocksworld. Valmeekam et al. (2023b) also assess the capabilities of pretrained LLMs on PDDL but their prompts include a general task description in NL and the PDDL domain in addition to the target problem and examples.

There is a lot of work being done on LLMs in the context of PDDL planning most of which is only remotely related to ours. For example, Rossetti et al. (2024) investigate learning per-domain generalizing policies by training transformer models on PDDL from scratch. Another line of research focuses on using LLMs to guide symbolic search (e.g. Hazra, Zuidberg Dos Martires, and De Raedt 2024; Hao et al. 2023; Zhou et al. 2024). Katz et al. (2024) and Silver et al. (2024) propose to let LLMs generate Python code to be used for deriving plans.

3 Converting PDDL into Natural Language

We contribute an automatic conversion, PDDL2NL, of PDDL domains and problems into NL descriptions. Our method first produces PDDL-to-NL mappings for objects, predicates, and actions, then joins these together into a prompt tasking the LLM to solve the described planning instance. To achieve this automatic conversion, we leverage an LLM (called 2NL-LLM, short for PDDL2NL-LLM), with few-shot prompting on generic examples (fixed once and used for any input PDDL) for each part of the conversion.

```
Your task is to translate given abstract descriptions into appropriate
natural language templates. The abstract descriptions include variables ...
starting with a question mark, such as ?x or ?object. Use the exact same
variable names. ... variable names need to be surrounded by curly brackets...
-----------------------------------------------------------------
Input: (planet ?ob)
Model: {?ob} is a planet

Input : (undertree ?obj)
Model: {?obj} is under the tree

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

Figure 2: 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).

Figure 2 illustrates the input prompt received by the 2NL-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 2NL-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. After the examples, we pass the actual predicate that we want to translate[4] to the LLM, i.e., we tell the 2NL-LLM which PDDL predicate we want to translate to NL

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

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

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

Table 1: Example PDDL-to-NL translation by the 2NL-LLM in the Logistics domain; predicates at the top; the "drive-truck" action at the bottom.

(e.g., we provide "(at ?x ?y)" as input) and expect the 2NL-LLM to return its NL description (e.g., "{?x} is at {?y}"). Table 1 (top) shows the NL descriptions of predicates generated in the Logistics domain. Note that the NL descriptions are constructed as *snippets* with placeholders for actual objects (e.g., "{?obj}") which are replaced later when constructing the final NL task description. Therefore, we require the LLM to generate snippets that include all variable names from the PDDL, each surrounded by brackets. We automatically check whether this condition is met and if not the 2NL-LLM is prompted once to revise the previous output.

NL snippets of actions are generated analogously using a similar prompt 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 snippets of the predicates generated by the 2NL-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 (e.g., Helmert 2009). We use the same four hand-crafted few-shot examples of actions independently of the target domain. As for predicates, snippets not matching the formal requirements are revised once.

Table 1 (bottom) shows an example translation of the "drive-truck" action from the Logistics domain. It illustrates two interesting characteristics of our NL snippets. First, the order of the arguments in the generated NL snippet can deviate from the order of the parameters in the input PDDL domain. The order of 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 2NL-LLM to prevent the LLM from inferring that the order needs to be identical. Second, the generated NL snippet 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 the extended version of this paper (Stein et al. 2025).

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

Figure 1b shows an excerpt from the NL *domain* description of the PDDL Logistics domain (Figure 1a). NL domain descriptions are designed to include the same information as the input PDDL. They start with the description of all possible actions ($\mathcal{A}^{\mathcal{PA}}$), followed by their preconditions ($\mathcal{A}^{\mathcal{PR}}$) and effects ($\mathcal{A}^{\mathcal{E}}$). If the domain is typed, a verbalization of the type hierarchy is added also ($\mathcal{T}$). 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. 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 1c and 1d) specify the goal condition ($\mathcal{G}$), available objects with their types ($\mathcal{O}$), and the initial state ($\mathcal{I}$).

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 1c) becomes "truck_0" (Figure 1d). 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 2NL-LLM. For example, the goal "(at p0 l0-0)" (Figure 1c) is converted into "package_0 is at location_0" and appended to "My goal is that in the end" (Figure 1d). 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,".

Our approach of letting the LLM generate NL snippets for the predicates and actions that are then used to compose the domain and problem descriptions in a rule-based way ensures that the logical structure of the PDDL definition is preserved. Therefore, our method guarantees that all action schemas are structurally correct in the NL description.

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) and LLM *policy* techniques (returning one action at a time). They both consist of three core components, namely P-LLM, T-LLM and a simulator (see

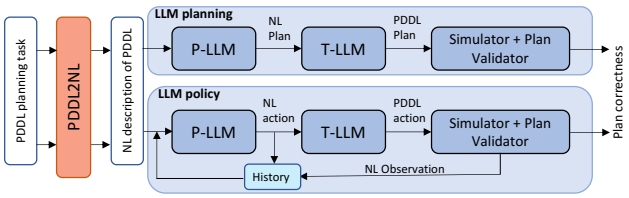

Figure 3: Overview of the set-up for the LLM plan generation and LLM action policy usage.

Figure 3). The NL description of the input PDDL task constructed using PDDL2NL 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 PDDL tasks. The simulator makes use of the plan validator VAL[5] 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. 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 (Wang et al. 2022; Shridhar et al. 2021; Yao et al. 2023; Lin et al. 2023), 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 PDDL2NL. 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 PDDL2NL. For the creation of grammatical negations of unsatisfied preconditions, we determine where to add negations based on the position of the auxiliary verb (if there is one) or of the head of the phrase as found by a dependency parser (Qi et al. 2020).

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

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

You are an assistant for giving instructions to successfully complete small tasks. ... Please instruct me how to complete my task. ... My task is to execute actions until reaching my goal.	1	You are an assistant for giving instructions to successfully complete small tasks. ... Please instruct me how to complete my task. ... My task is to execute actions until reaching my goal.
#NL DOMAIN DESCRIPTION#	2 $\mathcal{A}+\mathcal{T}$	#NL DOMAIN DESCRIPTION#
Please provide me a step-by-step instruction for how to complete my task. #THOUGHT INSTRUCTIONS# Please provide each step in a new line. Make sure to exactly follow the format of the provided example for your output...	3	Please instruct me how to complete my task. Please provide me only one single step at a time. #THOUGHT INSTRUCTIONS# 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'.
#FEW SHOT EXAMPLE#	4	#FEW SHOT EXAMPLE#
My goal is that in the end #NL GOAL# My current initial situation is as follows: #NL INITIAL STATE#	5 $\mathcal{G}$ $\mathcal{O}+\mathcal{I}$	My goal is that in the end #NL GOAL# My current initial situation is as follows: #NL INITIAL STATE#
#P-LLM OUTPUT: COMPLETE NL PLAN#	6	#1st P-LLM OUTPUT: NL ACTION# #1st NL OBSERVATION#

Figure 4: 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).

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 PDDL2NL and the objects from the current problem. Lastly, up to five pairs of an NL action snippet 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 Stein et al. 2025 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. (2023a).

P-LLM. The P-LLM component implements the actual action choice mechanism. Figure 4 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 domain (2) and specific instructions for the output of the individual LLM action-choice mechanisms (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 method and we discuss it in detail below. At the end, the goal and initial state of the target problem (5) is described (see complete prompts in Stein et al. 2025).

The action-choice instructions (part (3) 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 four action-choice mechanisms: two LLM planning techniques and two LLM

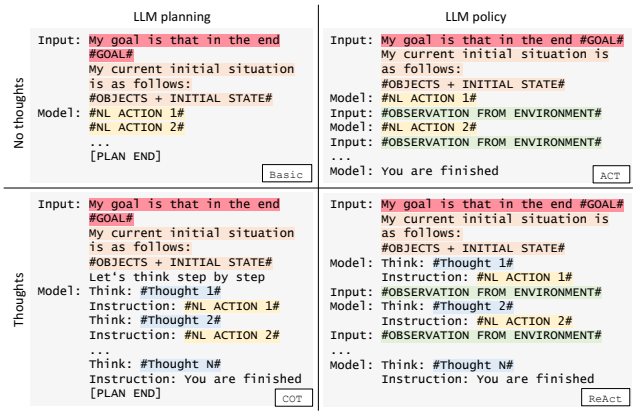

Figure 5: Structure of the few-shot examples for the four mechanisms.

policy techniques described in what follows.

LLM Planning

For LLM planning, the P-LLM predicts a complete sequence of NL actions, each in a separate line (see instructions (3) in Figure 4 (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. We include two LLM planning mechanisms: Basic and CoT.

Basic. Here, we prompt the P-LLM to generate a complete plan of NL actions. We follow Valmeekam et al. (2023b) and present each action of the example plan in a separate line and include a special tag to signal the end of the plan as shown in Figure 5 (top left).

CoT. Wei et al. (2022) 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 to satisfy. For example, a thought in Logistics 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 5, bottom left) and additional instructions in the prompt (see part (3) in Figure 4).

LLM Policies

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 3 (bottom)). In each step, the initial prompt for the P-LLM is extended by all its previous outputs and observations from the history buffer (see part (6) in Figure 4 (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.

We focus on two LLM policy techniques: ReAct and Act.

ReAct. ReAct combines CoT reasoning with information received from an environment. It was originally proposed for interactive decision-making tasks (Yao et al. 2023). At each step, the P-LLM predicts a thought and an action and receives an observation from the simulator (see Figure 5, bottom right). The complete output of the P-LLM including the thought and the observation are added to the history buffer.

Act. The Act mechanism works in the same way as ReAct but does not include reasoning thoughts (Figure 5, top right).

Note that all few-shot examples are generated automatically by converting one small example problem and its 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 5, bottom right), but with placeholders instead of actual thoughts. We manually create thoughts for one example problem from Logistics and use this as few-shot example when prompting an LLM to come up with appropriate thoughts to replace the placeholders for other problems and domains. For the CoT few-shot examples the observations get removed afterwards (see Stein et al. 2025).

5 Experiments

We evaluated all our action-choice mechanisms, `Bas` (Basic), `CoT`, `Act` and `ReA` (ReAct), on different kinds of input prompts, and against representatives of other planning mechanisms. We used GPT-4o as LLM for all evaluated methods. The LLM policies `Act` and `ReA` are not guaranteed to terminate and imposing a time limit is not an option here because we use GPT-4o via its API with high variability of response times. Therefore, we limit the number of steps for the LLM policies in each instance by using twice the length of the plan generated by satisficing greedy best-first search (GBFS) with the FF heuristic (Hoffmann and Nebel 2001).

All results for the LLM-based action choice methods are averaged over 5 runs; each run uses a different value of the "seed" parameter of the GPT-4o API. We compare our methods with PDDL2NL prompts against the following methods.

`Valm23`**:** We compare to the manually provided NL prompts designed by Valmeekam et al. (2023b) for the domains Blocksworld, Depots and Logistics.

`PDDL`**:** Following the prior work of Silver et al. (2022), we evaluate 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. Instead, the PDDL action is

extracted from the output of the P-LLM using a regular expression and is passed directly to the simulator. Additionally, the observations are modified to "The action #ACTION# was successfully executed." and "The action #ACTION# is not applicable because #UNSAT-PRE#." where #UNSAT-PRE# is a concatenation of the unsatisfied PDDL predicates, each followed by "is true" or "is false". 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 Stein et al. 2025 for example prompts). This is closer to the way in which Valmeekam et al. (2023b) test PDDL inputs as they, in contrast to Silver et al. (2022), include the PDDL domain description and also a short task description in natural language.

`Tpl`**:** As a baseline PDDL-to-NL translation, we use a simple template-based method for converting PDDL into NL prompts: Unary predicates "(p ?x)" are translated to snippets "{?x} is 'p"', higher-arity predicates "(p ?x1 ... ?xn)" are translated to "{?x1}, ..., and {?xn} are in relation 'p"', and PDDL actions "(act ?x1 ... ?xn)" are translated to "apply the action 'act' to object {?x1}, ..., and object {?xn}" where the word "object" is replaced with the parameter's type when it is specified. The composition of these snippets into the final prompt is done in the same way as in PDDL2NL. Since it is, again, not clear how to provide thoughts for this method, we use it only with `Bas` and `Act` mechanisms.

`rnd`**:** To test whether LLM-based methods carry any information at all, we use a simple random search (`rnd`) limited by the same number of steps as the LLM policy that, in every step, selects an applicable action uniformly at random. The results for `rnd` are averaged over 10 random seeds.

`BrFS`**,** `lmc`**,** `ff`**:** We use breadth-first search (`BrFS`) to get a sense of hardness of tasks and how LLM-based methods compare to a trivial symbolic baseline. We also use two strong optimal and satisficing planner baselines, namely A* with the LM-cut heuristic (`lmc`) (Helmert and Domshlak 2009), and GBFS with the FF heuristic (`ff`) (Hoffmann and Nebel 2001). These were run on Intel Xeon E5-2687W processors with 30 minutes and 8 GB time and memory limits.

We use two benchmark sets. First, we use 13 classical planning domains (see Table 2) including the three used by Valmeekam et al. (2023b) and the Beluga domain which was published after the training of GPT-4o (Eisenhut et al., 2024). For Beluga, we select the 21 smallest problem instances from Eisenhut et al. (2024). In the other domains, we generate 21 relatively small solvable instances with optimal plan lengths between 3 and 20. We select a small instance (short plan; see details in Stein et al. 2025) for the P-LLM few-shot example (plan or single action choice), and use the remaining 20 instances as our benchmarks.

Second, we also compare the best PDDL2NL action-choice mechanisms to the symbolic baselines (`rnd`, `BrFS`, `lmc`, `ff`) on a subset of IPC benchmarks where it is possible and feasible. First, we disregard domains with PDDL features unsupported by PDDL2NL (e.g., conditional effects, quantifiers); and since LLM-based methods can rarely scale beyond `ff` (as we show later), we remove instances that cannot be solved by `ff` within 30 minutes. Apart from this, the main limiting factor is feasibility in terms of monetary cost for GPT-4o calls. For 16 domains, the estimated cost (per domain!) was > 150USD; for those, we considered only the instances solved by BrFS, reducing the cost to ≤ 150USD for 15 domains. In addition to the monetary issues, we could not successfully run our method on Agricola, Folding, Parking, Richochet-Robots, Sokoban, Tetris, and Tidybot (e.g., in some cases PDDL2NL failed to generate NL snippets due to large numbers of action parameters). Overall, the resulting IPC benchmark set consists of 675 instances in 41 domains.

Domains	LLM-based Approaches								Symb. Baselines			
	PDDL2NL				Tpl		PDDL					
	Bas	CoT	Act	ReA	Bas	Act	Bas	Act	rnd	BrFS	lmc	ff
Block.	13	15	17	**18**	8	12	13	14	1	20	20	20
↳Valm23	15	14	17	18								
Logistics	5	10	16	**19**	2	7	6	15	0	20	20	20
↳Valm23	3	5	17	12								
Depot	0	5	5	13	0	5	0	3	0	20	20	20
↳Valm23	3	6	6	**15**								
Beluga	4	4	8	**9**	3	5	5	3	0	20	0	20
Ferry	7	12	14	**18**	0	13	8	17	0	20	20	20
Floortile	0	0	0	0	0	0	0	0	0	18	20	20
Goldm.	1	1	3	1	1	3	1	**4**	0	20	20	20
Grid	8	6	16	**18**	1	12	6	12	0	20	20	20
Grippers	9	17	17	**20**	10	**20**	12	19	0	20	20	20
Movie	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	3	20	20	20
Rovers	0	0	**18**	**18**	1	17	1	11	1	20	20	20
Satellite	14	16	**20**	**20**	11	18	14	18	0	20	20	20
Visitall	19	19	**20**	**20**	18	**20**	**20**	**20**	8	20	20	20
Σ (260)	100	125	174	**194**	75	152	106	156	13	258	240	260
Further scaled selected domains:												
Block.	3	3	12	**14**	0	6	1	4	0	12	19	20
Ferry	0	0	7	15	0	9	0	**17**	0	8	13	20
Gripper	17	12	**20**	19	16	**20**	16	**20**	0	10	8	20
Visitall	9	2	16	**18**	7	17	14	16	1	10	18	20
Σ (80)	29	17	55	**66**	23	52	31	57	1	40	58	80

Table 2: Number of solved tasks out of 20 per domain. "Valm23" rows show results of the manual encodings by Valmeekam et al. (2023a) in the respective domains. We show in **bold** the best LLM-based method. LLM-based results are averaged over 5 seeds, `rnd` over 10 seeds.

Comparison to `Valm23`**.** 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. The most significant difference can be observed for `CoT` and `ReA` action-choice mechanisms in Logistics where our method solves 5 and 7 more tasks, respectively. This is a little bit surprising result considering that the hand-crafted descriptions by Valmeekam et al. contain additional information that is not explicitly stated in PDDL (e.g., that all locations within a city are directly connected) which is thus not available in the NL descriptions automatically generated by PDDL2NL.

Comparison between PDDL2NL variants. 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., Valmeekam et al. 2023a; Liu et al. 2023). Adding reasoning thoughts (`CoT`) improves performance substantially overall, though the impact varies per domain and can also deteriorate

performance (namely in the Grid domain).

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 domains. This shows that the use of LLMs, not for plan generation, but as *a part of plan generation* works much better—in this 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 `ReA` yields another performance boost, consistently dominating coverage across all four LLM action choice mechanisms (with an exception in Goldminer) and achieving best LLM performance.

Comparison to `PDDL` and `Tpl`. `PDDL` performs slightly better than PDDL2NL for the `Bas` variant, whereas it is the other way around for `Act`. Nevertheless, `ReA` with PDDL2NL is clearly superior mainly because it uses natural sounding, intuitive thoughts in NL. It is not clear how we could obtain such thoughts for `PDDL`.

The simple template-based method (`Tpl`) works decently in some domains (e.g., Depot, Goldminer, or Visitall), but PDDL2NL generates NL task descriptions that are at least as good (and often better) in most domains. We think this is because LLMs "understand" language and therefore the NL descriptions generated by PDDL2NL are more naturally-sounding than the `Tpl` descriptions. In comparison to `PDDL`, `Tpl` is clearly the weaker method. It also seems that `Tpl` works well in domains where also `PDDL` works well, and whenever PDDL2NL works significantly better than `Tpl`, it also works better than `PDDL`.

We additionally ran experiments on a version of Blocksworld where the structure of the domain is preserved but the actions and predicates have only very generic names. In particular, we replaced all predicate names with "predicate1", ..., "predicate$N$" and action names with "action1", ..., "action$M$", where N and M are the number of predicates and actions in the PDDL file, respectively. This resulted in very generic, template-like, NL snippets produced by the LLM (e.g. "predicate3 is true", "perform action1 on an object {?obj}"). Running the `Bas` and `Act` approaches on the generated prompts yielded 0 coverage for both approaches.

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 the exception of Floortile where all LLM methods failed). 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: `ff` solves all tasks, `lmc` solves all tasks except for the Beluga domain, and even `BrFS` solves all tasks except for two in Floortile (the average runtime of `lmc`, `ff` and `BrFS` was 0.2, 0.1 and 6 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 `ReA` in many domains.

Domains	PDDL2NL		Symbolic Baselines			
	`CoT`	`ReA`	`rnd`	`BrFS`	`lmc`	`ff`
barman11/14 (10)	0	3	0	10	3	10
blocks00 (35)	3	22	0	21	28	35
childsnack14 (16)	6	15	0	0	0	16
gripper98 (19)	12	19	0	7	6	19
logistics98/00 (29)	1	28	0	12	21	29
movie98 (29)	29	29	0	29	29	29
rovers06 (6)	1	5	0	6	6	6
satellite02 (5)	1	4	0	5	5	5
transport08/11 (31)	3	23	0	18	19	31
visitall11/14 (13)	6	13	0	13	13	13
others (482 in 27 domains)	4	18	1	291	311	482
Σ (675)	66	179	1	412	441	675

Table 3: Number of solved tasks for selected IPC instance suites. "others": sum over IPC instance suites where both `CoT` and `ReA` solved less than 25% of instances.

Scaling experiments. To see how far the LLM action choice mechanisms can scale, we conduct more experiments with larger generated tasks. We focus on Blocksworld, Ferry, Grippers, and Visitall because `ReA` performs very well there and it is easy to scale these domains with a single parameter.

The scaled data for these 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 Stein et al. 2025 for more details). Then we run `BrFS` and `lmc` on each task with 30 minutes and 8 GB limits. Then 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. The bottom part of Table 2 shows the coverage on the scaled benchmark set.

`rnd` can solve only a single task in Visitall, confirming the vast superiority of LLM-based methods as much more informed. `BrFS` is much better, but is also challenged by the size of these tasks. It is outperformed by `ReA` in all domains. While `BrFS` is a very basic symbolic baseline, this provides additional evidence of `ReA`'s planning abilities. Indeed, remarkably, `ReA` outperforms the state-of-the-art optimal planner `lmc` in 2 domains (and matches its performance in 1). This is remarkable given that `ReA`, 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 `ReA`—in contrast to `lmc`—does not give any plan-optimality guarantee. The satisficing planner `ff`, which is comparable to `ReA` 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 (as in case of `Act` and `ReA`).

IPC benchmarks. We also conducted experiments on a subset of IPC domains with the best plan-generation (`CoT`)

and policy (ReA) methods (see Table 3). CoT and ReA solved less than 25% in 32 and 27 IPC instance suites, respectively, and 0 tasks in 29 and 20 instance suites, respectively. CoT solved all tasks solved by ff only in movie98, but ReA matched the performance of ff in movie98 and gripper98 (and 3 more suites limited to instances solved by BrFS), and solved only one less task in childsnack14 and logistics00 (and 2 more suites limited by BrFS). This shows that LLM-based methods are usually unable to scale to larger instances, but also that there are domains where ReA is able to achieve a good performance. In fact, it seems that ReA either works fairly well or not at all (with very few outliers).

Cost analysis. We additionally conduct a comparison of the cost for the different input prompt versions in terms of the total number of input tokens processed by the P-LLM on solved problem instances. In particular, we compute the average number of input tokens for each domain in Table 2, considering only those problem instances that were solved by Act for each of PDDL2NL, Tpl and PDDL. The sum of the per-domain averages for inputs generated by PDDL2NL is 816 732 tokens and hence cheaper than the Tpl approach with 1 064 573 tokens. In comparison to PDDL inputs, our approach is more costly (816 732 vs. 665 881), but by only up to a factor of 1.5 per domain.

We also compare our automatic approach with the human-generated input prompts from Valmeekam et al. (2023a). The sums of the average number of tokens for commonly solved instances of Act for PDDL2NL and Valm23 are 69 137 vs. 52 080, respectively. Hence, our automatic approach is more costly but only slightly and only up to a factor of 1.5 per domain. We provide the per-domain averages in the extended version of this paper (Stein et al. 2025).

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 (Valmeekam et al. 2023b) with LPG (Gerevini, Saetti, and Serina 2003)), 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.

Acknowledgments

The work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under the project number 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 under project number 471607914.

References

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

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

Eisenhut, J.; Schuler, X.; Fišer, D.; Höller, D.; Christakis, M.; and Hoffmann, J. 2024. New Fuzzing Biases for Action Policy Testing. *Proceedings of the International Conference on Automated Planning and Scheduling*, 34(1): 162–167.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through Stochastic Local Search and Temporal Action Graphs. *Journal of Artificial Intelligence Research (JAIR)*, 20: 239–290.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - The Planning Domain Definition Language.

Gupta, T.; and Kembhavi, A. 2023. Visual Programming: Compositional Visual Reasoning Without Training. In *Pro-*

ceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 14953–14962.

Hao, S.; Gu, Y.; Ma, H.; Hong, J.; Wang, Z.; Wang, D.; and Hu, Z. 2023. Reasoning with Language Model is Planning with World Model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 8154–8173.

Hazra, R.; Zuidberg Dos Martires, P.; and De Raedt, L. 2024. SayCanPay: Heuristic Planning with Large Language Models Using Learnable Domain Knowledge. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(18): 20123–20133.

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

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS*. AAAI.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Katz, M.; Kokel, H.; Srinivas, K.; and Sohrabi, S. 2024. Thought of Search: Planning with Language Models Through The Lens of Efficiency. In *Advances in Neural Information Processing Systems*, volume 37, 138491–138568.

Khot, T.; Khashabi, D.; Richardson, K.; Clark, P.; and Sabharwal, A. 2021. 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*, 1264–1279.

Lin, B. Y.; Fu, Y.; Yang, K.; Brahman, F.; Huang, S.; Bhagavatula, C.; Ammanabrolu, P.; Choi, Y.; and Ren, X. 2023. SwiftSage: A Generative Agent with Fast and Slow Thinking for Complex Interactive Tasks. In *Advances in Neural Information Processing Systems*, volume 36, 23813–23825.

Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; Biswas, J.; and Stone, P. 2023. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. Preprint arXiv:2304.11477.

OpenAI. 2024. Hello GPT-4o. Accessed: 2024-10-28.

Prasad, A.; Koller, A.; Hartmann, M.; Clark, P.; Sabharwal, A.; Bansal, M.; and Khot, T. 2024. ADaPT: As-Needed Decomposition and Planning with Language Models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, 4226–4252.

Qi, P.; Zhang, Y.; Zhang, Y.; Bolton, J.; and Manning, C. D. 2020. Stanza: A Python Natural Language Processing Toolkit for Many Human Languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Rossetti, N.; Tummolo, M.; Gerevini, A. E.; Putelli, L.; Serina, I.; Chiari, M.; and Olivato, M. 2024. Learning General Policies for Planning through GPT Models. *Proceedings of the International Conference on Automated Planning and Scheduling*, 34(1): 500–508.

Shridhar, M.; Yuan, X.; Cote, M.-A.; Bisk, Y.; Trischler, A.; and Hausknecht, M. 2021. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *International Conference on Learning Representations*.

Silver, T.; Dan, S.; Srinivas, K.; Tenenbaum, J. B.; Kaelbling, L. P.; and Katz, M. 2024. Generalized Planning in PDDL Domains with Pretrained Large Language Models. In *38th AAAI Conference on Artificial Intelligence (AAAI'24)*.

Silver, T.; Hariprasad, V.; Shuttleworth, R. S.; Kumar, N.; Lozano-Pérez, T.; and Kaelbling, L. P. 2022. PDDL Planning with Pretrained Large Language Models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Stein, K.; Fišer, D.; Hoffmann, J.; and Koller, A. 2025. Automating the Generation of Prompts for LLM-based Action Choice in PDDL Planning. arXiv:2311.09830.

Valmeekam, K.; Marquez, M.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2023a. PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Valmeekam, K.; Marquez, M.; Sreedharan, S.; and Kambhampati, S. 2023b. On the Planning Abilities of Large Language Models - A Critical Investigation. In *Advances in Neural Information Processing Systems*, 75993–76005.

Wang, R.; Jansen, P.; Côté, M.-A.; and Ammanabrolu, P. 2022. ScienceWorld: Is your Agent Smarter than a 5th Grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 11279–11298.

Wang, Z.; Cai, S.; Chen, G.; Liu, A.; Ma, X. S.; and Liang, Y. 2023. Describe, Explain, Plan and Select: Interactive Planning with LLMs Enables Open-World Multi-Task Agents. In *Advances in Neural Information Processing Systems*, volume 36, 34153–34189.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q. V.; and Zhou, D. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, volume 35, 24824–24837.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K. R.; and Cao, Y. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.

Zhou, A.; Yan, K.; Shlapentokh-Rothman, M.; Wang, H.; and Wang, Y.-X. 2024. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24.

Zhu, X.; Chen, Y.; Tian, H.; Tao, C.; Su, W.; Yang, C.; Huang, G.; Li, B.; Lu, L.; Wang, X.; Qiao, Y.; Zhang, Z.; and Dai, J. 2023. 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.