# PROC2PDDL: Open-Domain Planning Representations from Texts

**Tianyi Zhang**[1*] **Li Zhang**[1*] **Zhaoyi Hou**[3]
**Ziyu Wang**[1] **Yuling Gu**[2] **Peter Clark**[2]
**Chris Callison-Burch**[1] **Niket Tandon**[2]
[1]University of Pennsylvania [2]Allen Institute for Artificial Intelligence
[3]University of Pittsburg
{zty|zharry|ccb}@upenn.edu

## Abstract

Planning in a text-based environment continues to be a significant challenge for AI systems. Recent approaches have utilized language models to predict planning domain definitions (e.g., PDDL) but have only been evaluated in closed-domain simulated environments. To address this, we present PROC2PDDL, the first dataset containing open-domain procedural texts paired with expert-annotated PDDL representations. Using this dataset, we evaluate the task of predicting domain actions (parameters, preconditions, and effects). We experiment with various large language models (LLMs) and prompting mechanisms, including a novel instruction inspired by the zone of proximal development (ZPD), which reconstructs the task as incremental basic skills. Our results demonstrate that PROC2PDDL is highly challenging for end-to-end LLMs, with GPT-3.5's success rate close to 0% and GPT-4o's 38%. With ZPD instructions, GPT-4o's success rate increases to 45%, outperforming regular chain-of-thought prompting's 34%. Our analysis systematically examines both syntactic and semantic errors, providing insights into the strengths and weaknesses of language models in generating domain-specific programs.[1]

## 1 Introduction

Planning is the task of finding a sequence of actions to achieve a goal in a given environment (Fikes and Nilsson, 1971; LaValle, 2006). In real life, the environment is often described with natural language texts. To enable text-based, automated planning, recent work has used language models (LMs) to *generate plans* (Valmeekam et al., 2023a; Stein and Koller, 2023). However, this approach is found to fall short with regard to both performance and interpretability (Valmeekam et al., 2023c,b). Alternatively, another recent line of worked has instead used LMs to *translate* the natural language description of environments to planning domain definition language (PDDL) (Ghallab et al., 1998). This symbolic representation can then be solved by a planner in a plan (Collins et al., 2022; Lyu et al., 2023; Liu et al., 2023; Xie et al., 2023; Wong et al., 2023). Despite of the success of such a neurosymbolic method, all the above work has only been evaluated in **closed-domains** simulated environments such as a household (e.g., ALFRED (Shridhar et al., 2020)) or discrete object placement (e.g., BlocksWorld (Valmeekam et al., 2024)) (as shown in Table 1).

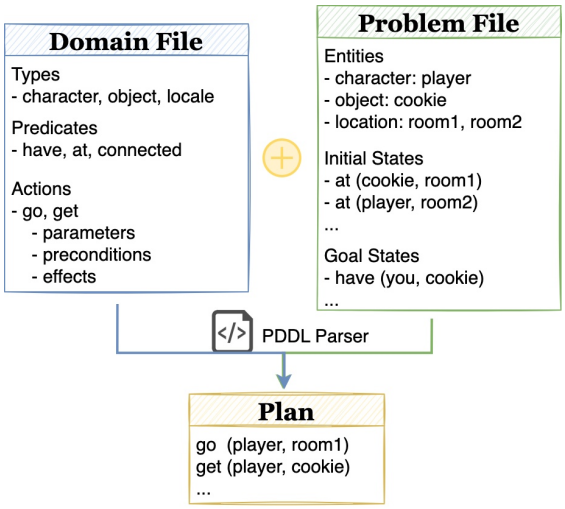


Figure 1: A PDDL solver produces a plan based on a minimal domain file and problem file. Previous work assumes the domain file as given, while we predict the action definitions in the domain file.

To enable **open-domain**, text-based planning, we propose PROC2PDDL, a dataset to evaluate models' ability to generate PDDL given procedural texts. PROC2PDDL consists of 27 pairs of open-domain procedures and PDDL representations. Each PDDL representation include a domain file $\mathbb{DF}$ that models the types, predicates, and actions, and a problem file $\mathbb{PF}$ that models the entities, initial states, and goal states, as illustrated in Figure 1. Because PROC2PDDL is not bound

[*]Equal contribution.
[1]Our resources can be found at https://github.com/zharry29/proc2pddl.



*Proceedings of the 2nd Workshop on Natural Language Reasoning and Structured Explanations (@ACL 2024)*, pages 13–24

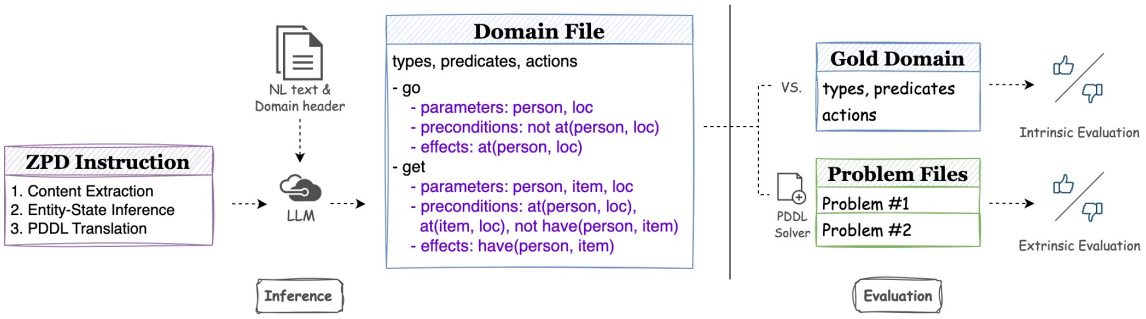


Figure 2: Our formulation of the $\mathbb{DF}$ action prediction task is as follows: given a natural language procedure text and a domain file header, a language model (LM) follows Zone of Proximal Development (ZPD) instructions in three sequential skills to predict domain actions, including parameters, preconditions, and effects. During evaluation, the predicted $\mathbb{DF}$ is compared to a gold reference and used to solve corresponding $\mathbb{PF}$s.

to any simulation, the PDDL representations are manually annotated by experts trained on this task to ensure validity, resulting in 27 domain files and 95 problem files.

Using this dataset, we study the task of action modeling (Lindsay et al., 2017) formulated as follows. The input is some relevant natural language texts and the *header* of a $\mathbb{DF}$ (i.e., types, predicates, and names of actions). Based on a ZPD instruction, the output is the *domain actions* in the $\mathbb{DF}$ (i.e., parameters, preconditions, and effects). During evaluation, the predicted $\mathbb{DF}$ is 1) compared to a ground-truth $\mathbb{DF}$ as intrinsic evaluation, and 2) provided to a PDDL solver with ground-truth $\mathbb{PF}$s for the existence and correctness of plans as extrinsic evaluation. Our system is delineated in Figure 2. In this formulation, our assumption of the $\mathbb{DF}$ header is necessary to ensure the consistency of semantics between the $\mathbb{DF}$ and the $\mathbb{PF}$ for evaluation. It is also empirically motivated; for example, a kitchen robot may have access to the types like 'ingredients' and predicates like 'diced' via some information extraction system given descriptive texts, but it may still need to predict, for "swinging a knife", the precondition that it is only safe to do so to the 'ingredients' and the effect that they will become 'diced'.

Through our experiment, we show that the task of action modeling in PROC2PDDL is highly challenging to state-of-the-art LMs, where GPT-3.5 almost fails completely, GPT-4 can only generate exactly matching $\mathbb{DF}$s 16% of the time and solvable $\mathbb{PF}$s 33% of the time, and GPT-4o demonstrate 18% $\mathbb{DF}$s accuracy and 37% $\mathbb{PF}$s solving rate. By devising a ZPD instruction that prompt LMs to modularly generate PDDL through extraction-inference-translation approach, we improve action accuracy by 3% and problem solving by 2-7% . In our analysis, the syntactic errors indicate LMs' weakness in generating low-resource and domain-specific programming languages (Cassano et al., 2023) like PDDL, while the semantic errors suggest LMs' inaccuracies to reason about actions and environments.

| | #$\mathbb{DF}$ | Datasets |
|---|---|---|
| Ours | 27 | PROC2PDDL |
| (Wong et al., 2023) | 2 | MineCraft, ALFRED |
| (Lyu et al., 2023) | 1 | SayCan |
| (Xie et al., 2023) | 2 | Blocksworld, ALFRED |
| (Liu et al., 2023) | 7 | Blocksworld, etc. |
| (Huang et al., 2023) | 1 | Tabletop |
| (Huang et al., 2022) | 1 | VirtualHome |
| (Silver et al., 2022) | 18 | Blocksworld, etc. |
| (Valmeekam et al., 2022) | 2 | Blocksworld, Logistics |

Table 1: Our work proposes and evaluates models using PROC2PDDL which is open-domain and based on procedural texts, while past work has relied on closed-domain benchmarks which can be expressed with a singular $\mathbb{DF}$ with a fixed set of actions, based on some simulation.

## 2 Task Formulation

The task of predicting a planning domain definition in a text-based environment can be seen as translating natural language texts to PDDL symbolic language, which consists of a domain file ($\mathbb{DF}$) and one or more problem files ($\mathbb{PF}$s).

A $\mathbb{DF}$ defines all actions in the environment:

- parameters (e.g., water, pot) as a list of typed variables
- preconditions (e.g., water and pot belongs to player; water is not treated) as a conjunctive normal form of predicates
- effect (e.g., water is treated) as a conjunctive normal form of predicates

A $\mathbb{PF}$ defines the initial and goal environments:

- initial states (e.g., bucket is empty)
- goal states (e.g., bucket is filled with rainwater; rainwater is treated)

We say that a $\mathbb{DF}$ and a $\mathbb{PF}$ can be solved if there exists a sequence of actions $A_1, \ldots, A_n$ that results in a transition from the initial state to the goal state.

Traditionally, the task of text-based PDDL generation involves predicting $\mathbb{PF}$ based on text $\mathbb{T}$, where a successfully generated $\mathbb{PF}$ can be solved by the predefined $\mathbb{DF}$.

In this paper, we address an alternative formulation, action modeling ($A$), in which the generated $\mathbb{DF}$, given text $\mathbb{T}$ and the domain header $H$[2], is capable of producing plans for $\mathbb{PF}$s.

## 3 Dataset

We introduce the PROC2PDDL dataset of 27 different $\mathbb{T}$-$\mathbb{DF}$-$\mathbb{PF}$s tuples, drawing procedural texts from wikiHow articles of various topics (see Appendix A). A class of graduate students in a U.S. university with prior knowledge of PDDL are each given a wikiHow article and annotate a $\mathbb{DF}$ and multiple corresponding $\mathbb{PF}$s from the article, each with a gold plan to solve it. On average, there are 13.33 defined actions in a $\mathbb{DF}$ and 8.07 instantiated actions in a gold plan. In this work, all our data is used for evaluation, as all our methods are without task specific model training. Some sample data of PROC2PDDL can be found in Appendix B.

## 4 Methodology

We first introduce a novel prompt design option, ZPD, and then discuss the choices of text format ($\mathbb{T}$), which can range from 10 to 2,000 tokens and influence the selection of LMs.

### 4.1 ZPD Prompt Design

To predict domain actions $A$ based on relevant $\mathbb{T}$ and the header $H$, we prompt an LM in zero-shot or few-shot instructions. Our instruction employs Zone of Proximal Development (ZPD) theory proposed for human learning (Vygotsky and Cole, 1978), which is a variant of the chain-of-thought (CoT) approach. In typical CoT, a task is decomposed into several constituents (steps), i.e., parameters, precondition, and effect. In contrast, according to ZPD, the complex PROC2PDDL task is decomposed into atomic **skills**: 1) *extracting* the relevant *description* of an action; 2) *extracting* and *inferring* the incorporated *entities* and their *state changes*; and 3) *translating* the entity-state changes to accessible PDDL predicates. Next, we establish the relationships between these atomic skills: to perform the task, each skill is a prerequisite for the next. Finally, we explicitly instruct the LMs to incrementally perform the three basic skills, leading to the successful completion of the PROC2PDDL task (the prompt can be found in Appendix D):

1. Extraction: describe each action, including the expected preconditions and effects;
2. Inference: list the involved entities and their state changes;
3. Translation: based on the information above, convert $\mathbb{T}$ to PDDL.

[2]The domain header includes types, predicates, and names of actions in $\mathbb{DF}$. As the information specified by $H$ is guaranteed to be consistent with that of the $\mathbb{PF}$s, the evaluation is well-defined.

| Model % | Intrinsic action acc. | Extrinsic $\mathbb{PF}$ solve |
|---|---|---|
| `gpt-3.5` | 0.2 | 1.0 |
| `gpt-4` | 15.9 | 33.7 |
| `+ CoT` | 9.3 | 21.1 |
| `+ ZPD` | **18.1** | **35.8** |
| `+ ZPD, 3 shot` | 11.9 | 23.2 |
| `gpt-4o` | 18.2 | 37.9 |
| `+ CoT` | 19.5 | 33.7 |
| `+ ZPD` | **21.4** | **45.3** |
| `+ ZPD, 3 shot` | 20.3 | 40.0 |
| gold | 100 | 100 |

Table 2: The intrinsic and extrinsic evaluation results for all main models. `gpt-4(o)` demonstrates non-trivial performance. With a ZPD instruction, the performance improves consistently.

| Model % | Parameter | Precondition | Effect |
|---|---|---|---|
| `gpt-4` | 36.7 | 31.1 | **53.0** |
| `+ CoT` | 29.7 | 25 | 54.7 |
| `+ ZPD` | **42.2** | 29.7 | 48.1 |
| `gpt-4o` | 45.1 | 31.1 | **62.5** |
| `+ CoT` | 52.4 | 34.2 | 54.1 |
| `+ ZPD` | **53.5** | **40.1** | 53.5 |

Table 3: The generation accuracy of each component in actions has been evaluated. The ZPD instruction clearly aids in identifying implicit parameters (entities). Predicting preconditions is more challenging than predicting effects, as it requires a greater depth of implicit knowledge of entity states.

### 4.2 Choice of Input Text

We also consider the following choices of wikiHow text as $\mathbb{T}$.

**Prompt without text (w/o $\mathbb{T}$)** is an ablation baseline where the model predicts $A$ solely based on $H$. Naturally, none of the three aforementioned stages are involved in this prompt condition.

| Model % | Intrinsic action acc. | Extrinsic $\mathbb{PF}$ solve |
|---|---|---|
| w/o $\mathbb{T}$ (baseline) | 13.7 | 26.3 |
| $\mathbb{T}$ =sum | 15.9 | 33.7 |
| $\mathbb{T}$ =sum, ZPD | **18.1** | **35.8** |
| $\mathbb{T}$ =map | 11.8 | 13.7 |
| $\mathbb{T}$ =map, ZPD | 8.9 | 26.3 |
| $\mathbb{T}$ =rel | 11.6 | 27.4 |
| $\mathbb{T}$ =rel, ZPD | 12.2 | 21.1 |
| $\mathbb{T}$ =all | 12.1 | 28.4 |
| $\mathbb{T}$ =all, ZPD | 12.1 | 31.6 |

Table 4: Performance of GPT-4 using different portions of text $\mathbb{T}$. Metrics include action-wide accuracy and the proportion of $\mathbb{PF}$s that can be solved.

**Prompt with text (w/ $\mathbb{T}$)** additionally provides the model with four different portions of $\mathbb{T}$, involving the three aforementioned stages, as follows:
**($\mathbb{T}$ = all)**: All steps in a wikiHow article.
**($\mathbb{T}$ = rel)**: In PROC2PDDL, each wikiHow article consists of step paragraphs that may or may not be used in defining the actions in the $\mathbb{DF}$. Hence, a mapping between actions and steps is also annotated. This context includes relevant steps to all actions in a $\mathbb{DF}$. (e.g., Step 1. Find fresh water... Step 2. Collect food... Step 7. Set up camp...)
**($\mathbb{T}$ = map)**: Each action is mapped with steps based on the annotated mapping in PROC2PDDL.
(e.g., clean_water: Step 1. Find fresh water...)
**($\mathbb{T}$ = sum)**: An one-line summary of each action annotated in PROC2PDDL.
(e.g., clean_water; boil water to clean it)
The four prompts are increasingly general. Distinguishing from the required skills, the full text condition demands accurate information extraction, while the text summary clearly defines the action but requires the model's robust ability to infer implicit entity states. All prompts request an exact translation.

### 4.3 Experiments

We conducted experiments with three large language models[3]: GPT-3.5-turbo-16k, GPT-4-32k (dated June 2023), and GPT-4o. For GPT-4-32k, we used a maximum token limit of 10,000. GPT-3.5-turbo-16k and GPT-4o were tested with theirs default hyperparameters. The few-shot examples can be found in Appendix C.

[3]Due to the need for very long input and output, the choice of open-source models is limited. We are in progress of implementing `Mixtral-8x7B`.

## 5 Evaluation and Analysis

Now that a model generates the parameters, preconditions, and effects for each action, we have a complete $\mathbb{DF}$. We evaluate it in two ways (Figure 2). *Intrinsically*, we semantically compare the predicted $A$ with the ground-truth provided by our PROC2PDDL and report an action-wide accuracy. Equivalence of two action definitions does not depend on the naming of variables nor on the order within conjunctions (detailed in Appendix E). *Extrinsically*, to measure actions' coherence, a BFS-based PDDL solver[4] attempts to solve ground-truth $\mathbb{PF}$s with the predicted $\mathbb{DF}$ and a success rate is reported. An unsolved $\mathbb{PF}$ is caused by (1.) no plan can be found, or (2.) the solver runs for more than 30 seconds, or (3.) the solver returns an error (usually a syntax error in the generated PDDL).

The intrinsic and extrinsic results are shown in Table 2. `gpt-3.5-turbo` which achieves impressive performance on many tasks has a close-to-zero performance. In contrast, `gpt-4` performs significantly better with 18% action prediction accuracy and 36% solve rate of $\mathbb{PF}$s. The most advanced `gpt-4o` presents the highest performance, with 21% action accuracy and 45% $\mathbb{PF}$s solving rate. Still, the performance is far worse than ideal, showing that even a simplified open-domain planning formulation is challenging to state-of-the-art LMs.

**ZPD Instruction Analysis**
ZPD is helpful in each setting since it explicitly spells out many implicit entities and state changes in the inference stage which are critical to predicting parameters. In most situations, the model summarizes the action and extracts the entity states correctly, though sometimes missing a few implicit entities. However, ZPD's bottleneck lies in the translation stage, during which there are mainly three types of errors.

1. mismatched predicates: the model uses `(at ?loc ?item)` instead of `(inventory ?item)`;
2. hallucinated predicates: the model creates a new predicate `(soaked ?item)` while neglecting the existing `(submerged ?item)`;
3. complicated predicates: the model adds unnecessary predicates `(inventory ?submerged_item - item)` when already has `(inventory ?item)`.

[4]https://github.com/pucrs-automated-planning/pddl-parser

| | Unsolved | | | Solved | |
|---|---|---|---|---|---|
| | Syntactic Error | Bad Action | Good Action | Bad Plan | Good Plan |
| gpt-4 | 3 | 7 | 2 | 0 | 3 |

Table 5: A small-sample inspection shows that models make both syntactic and semantic errors.

To address these, we leave to future work to demonstrate and standardize the translation process by clearly describing all necessary entity-state change and encouraging the model to compare and strictly match the given predicates. Finer-grained evaluation results are shown in Table 3 to tease out the performance regarding such component within an action. It is clear that the LM is worse at predicting preconditions than at predicting effects. This is understandable as procedural texts like wikiHow tend to be less explicit about predictions than about effects (e.g., from 'bake for 10 minutes' it is obvious that the food will be baked, but it is unclear what state it had been in).

**Text Format Analysis**

As shown in Table 4, in w/o $\mathbb{T}$ setting, fully relying on its implicit knowledge, the model is already capable of inferring PDDL syntactically and semantically. In w/ $\mathbb{T}$ settings, our model shows an 'U' performance in terms of the text length. Using a sentence-long description for each action ($\mathbb{T}$ = `sum`) provided by PROC2PDDL, the model achieves the best performance among all, showing a strong deduction ability with the limited but precise NL input. The $\mathbb{T}$ = `all` setting ensues, which requires the most extraction rather than inference. In contrast, the middle ones ($\mathbb{T}$ = `rel/map`) with decreasing signal-to-noise ratio lead to worse results, indicating its shortage of extraction-inference trade-off. The signals contain both the described entity states and step relations, explicitly and implicitly. This shortage may come less from the entity states *(e.g., fish, spear in hunt_fish)*, but more from the relation between actions *(e.g., make_spear to hunt_fish)* which may be expressed in the $\mathbb{T}$ = `sum` and `all` settings.

**Case Analysis**

To provide deeper insights into model performance, we manually inspect the model output of `gpt-4` on all 6 examples (15 $\mathbb{PF}$s) in the development set. We consider the following scenarios.

**Unsolved** Whenever the predicted $\mathbb{DF}$ cannot solve a $\mathbb{PF}$, either a syntactic or a semantic error has occurred. For a **syntactic error**, the output may contain illegal expressions that cannot be parsed. For example, `(inventory ?player (clean ?strips))` is unacceptable because the arguments to a predicate must be atomic types, not another predicate. For a **semantic error** (namely, a 'bad action'), we identify the first problematic action that differs with the ground-truth. For example, if the action `cut_plant` misses a critical effect of `(inventory ?player ?stalk)`, then other actions such as `graft_stalk` requiring it cannot be executed. At times, there could be false negatives where the predicted action definitions are in fact reasonable but still cannot lead to a solution (namely, a 'good action').

**Solved** Even when the predicted $\mathbb{DF}$ solves a $\mathbb{PF}$, the plan may be different from the gold plan. It is naturally possible that the predicted plan is a fluke made possible by under-specified preconditions or over-exaggerated effects, as well as loopholes in the $\mathbb{PF}$ leading to unreasonable shortcuts. For the example in Figure 1, a model could *cheat* by defining the action `get` by not requiring the person and object to be in the same location; thus, the predicted plan would unreasonably omit the action go. However, at times, the predicted plan could also be a reasonable alternative.

The statistics of these errors are shown in Table 5. When no solution can be found, true negative is highly likely as the model indeed makes aforementioned mistakes during action prediction. When some solution is found, false positive is still possible as the predicted plan may be unreasonable. See attached materials for a complete error analysis of these examples. Our aforementioned future pipeline that separates summarization and translation would likely mitigate these errors.

## 6 Conclusion

We present PROC2PDDL, the first open-domain dataset that juxtaposes natural language and planning domain definition language. Our experiments show that ZPD instructions facilitate LMs' performance, while still find it challenging to translate the precondition and effects of actions. We hope our instruction design, evaluations and dataset help future progress towards integrating the best of LM and formal planning.

## 7 Limitations

Any planning language, including PDDL which we consider in this work, is an approximation of planning in the real world and cannot accurately reflect its complexity. Due to the consideration for simplicity in the annotation process, we use the primitive version of PDDLs, with restricted expressions and syntax, instead of newer versions of the planning language which extend its syntax in a variety of way.

Annotating PROC2PDDL is extremely costly as it requires knowledge of PDDL and much effort to translate procedural texts to PDDL. Thus, our dataset is relatively small with a limited range of topics. Due to the highly complex and subjective nature of the annotation process, each annotated example may reflect idiosyncratic though processes and biases of the individual annotator.

As with many similar works, there is a known gap between high-level planning such as ours (with high-level actions like "boil") and the actions used by present-day robots (with low-level motor functions like "move"). However, like similar works, we believe our efforts can see more practical application in the near future.

Our modeling efforts so far have mainly considered options of zero-shot prompting. There of course exists many other approaches including the few-shot setting, fine-tuning, and the model distillaion paradigm, which we plan to experiment with in the future. Moreover, our evaluation is imperfect in that even a well-annotated DF-PF pair might have multiple successful plans. Manual inspection is still necessary to accurately gauge models.

## Acknowledgements

This research is supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the HIATUS Program contract #2022-22072200005. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

## References

Federico Cassano, John Gouwar, Francesca Lucchetti, Claire Schlesinger, Carolyn Jane Anderson, Michael Greenberg, Abhinav Jangda, and Arjun Guha. 2023. Knowledge transfer from high-resource to low-resource programming languages for code llms. arXiv preprint arXiv:2308.09895.

Katherine M. Collins, Catherine Wong, Jiahai Feng, Megan Wei, and Joshua B. Tenenbaum. 2022. Structured, flexible, and robust: benchmarking and improving large language models towards more human-like behavior in out-of-distribution reasoning tasks.

Richard E Fikes and Nils J Nilsson. 1971. Strips: A new approach to the application of theorem proving to problem solving. Artificial intelligence, 2(3-4):189–208.

Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. 1998. PDDL - the planning domain definition language. Technical Report "CVC TR-98-003/DSC TR-1165", Yale Center for Computational Vision and Control.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In International Conference on Machine Learning, pages 9118–9147. PMLR.

Wenlong Huang, Fei Xia, Dhruv Shah, Danny Driess, Andy Zeng, Yao Lu, Pete Florence, Igor Mordatch, Sergey Levine, Karol Hausman, et al. 2023. Grounded decoding: Guiding text generation with grounded models for robot control. arXiv preprint arXiv:2303.00855.

Steven M LaValle. 2006. Planning algorithms. Cambridge university press.

Alan Lindsay, Jonathon Read, Joao Ferreira, Thomas Hayton, Julie Porteous, and Peter Gregory. 2017. Framer: Planning models from natural language action descriptions. In Proceedings of the International Conference on Automated Planning and Scheduling, volume 27, pages 434–442.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+p: Empowering large language models with optimal planning proficiency.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 10740–10749.

Tom Silver, Varun Hariprasad, Reece S Shuttleworth, Nishanth Kumar, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2022. Pddl planning with pretrained large language models. In NeurIPS 2022 Foundation Models for Decision Making Workshop.

Katharina Stein and Alexander Koller. 2023. Autoplanbench:: Automatically generating benchmarks for llm planners from pddl. arXiv preprint arXiv:2311.09830.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2023a. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2024. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. Advances in Neural Information Processing Systems, 36.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023b. On the planning abilities of large language models–a critical investigation. arXiv preprint arXiv:2305.15771.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2022. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). arXiv preprint arXiv:2206.10498.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2023c. Large language models still can't plan (a benchmark for llms on planning and reasoning about change).

Lev Semenovich Vygotsky and Michael Cole. 1978. Mind in society: Development of higher psychological processes. Harvard university press.

Lionel Wong, Jiayuan Mao, Pratyusha Sharma, Zachary S Siegel, Jiahai Feng, Noa Korneev, Joshua B Tenenbaum, and Jacob Andreas. 2023. Learning adaptive planning representations with natural language guidance. arXiv preprint arXiv:2312.08566.

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

## A Topics

Below are a list of the titles of wikiHow articles in PROC2PDDL, chosen per the requirement of a gruaduate-level university class.

- create secret society
- throw an anime party
- open a coconut
- calculate pi
- hack
- get out of quicksand
- make a detective kit
- lock picking
- make papyrus
- survive on a desert island
- survive in the jungle
- survive a war
- survive a comet hitting earth
- survive a nuclear attack
- survive in the woods
- survive deserted island
- survive shark attack
- survive emp attack

Each topic may have one or more annotated $\mathbb{DF}$s representing different domains. The homogeneity of the last 7 topics is due to the class' topic of interactive fictions.

## B Sample Data: $\mathbb{T}$, $\mathbb{DF}$, and $\mathbb{PF}$

To exemplify PROC2PDDL, below is an example procedural text $\mathbb{T}$ titled 'survive in the jungle', up to the third step, truncating the rest.

```
1. Collect rainfall from leaves and bamboo
    stalks. Look for large leaves that collect
    rainfall and bend them into a funnel to pour
     the water into a bottle or straight into
    your mouth. Bend bamboo stalks to let the
    water that collects in the compartments flow
     out into a container or break the bamboo
    compartment off at the line that goes across
     the stalk to use it as a water bottle. You
    could also look for rock formations that
    form natural pools and collect rainwater,
    but it is best to do this after a fresh
    rainfall to avoid pools that have been
    sitting for a long time and may be
    contaminated with bacteria. If you don't
    have a water bottle or other container to
    collect water, try to find other natural
    containers in the jungle such as a coconut
    shell or piece of wood shaped like a bowl.
    You can also leave these items out when it
    rains to collect the fresh water.

2. Boil water from streams to kill any bacteria.
     Look for running streams to find fresh
    water. Filter out any particles through a
    sock, shirt, or other fabric, then start a
    fire and boil the water to kill bacteria
    that can make you sick. If you don't have a
    pot to boil water in, then you can use a tin
     can, single-walled stainless steel water
    bottle, or any other metal container. If you
     have no way of making a fire or boiling the
     water, then you should avoid drinking water
     from streams. It can be contaminated with
    many types of bacteria from animals that
    will make you very sick. Always avoid
```

```
    drinking water from stagnant pools as the
    water is likely contaminated.

3. Make a solar water still with a container and
     a plastic sheet. Dig a hole in an area that
     receives at least some direct sunlight and
    put a container, such as a water bottle or
    can, in the middle of the hole. Fill the
    space between the sides of the hole and the
    container with wet leaves. Place a plastic
    sheet over the top of the hole and put rocks
     or other heavy objects around the edges to
    hold it in place. Put a small stone in the
    middle of the sheet above the container. The
     plastic sheet will accumulate condensation
    that will drip down the underside of the
    sheet and into the container. This water is
    distilled and safe to drink. You can use
    natural containers such as bamboo or a
    coconut shell if you don't have a bottle or
    can. A solar still does not collect large
    amounts of water. It should be used as a
    supplemental source of water rather than a
    primary source.

......
```

Below is a sample annotated $\mathbb{DF}$ of the above:

```
(define (domain survive_in_the_jungle)
   (:requirements :strips :typing)
   (:types
       stone wood bamboo_container water fire
            sos_sign fruit - item
       basecamp - location
       ill dehydrated hungry - condition
       player
       direction
   )

   (:predicates
      (has_bamboo ?loc - location) ; this
          location has bamboo to create a
          container
      (has_rainfall ?loc - location) ; this
          location has received rainfall to
          collect water
      (has_fruit ?loc - location) ; this location
           has fruits to pick
      (treated ?water - water) ; True if the
          water has been decontaimated by
          boiling it
      (is ?c - condition ?p - player) ; True if
          the player is under the specified
          condition
      (at ?obj - object ?loc - location) ; an
          object is at a location
      (inventory ?player ?item) ; an item is in
          the player's inventory
      (connected ?loc1 - location ?dir -
          direction ?loc2 - location) ; location
           1 is connected to location 2 in the
          direction
      (blocked ?loc1 - location ?dir - direction
          ?loc2 - location) ; the connection
          between location 1 and 2 in currently
          blocked
   )

   (:action go ; navigate to an adjacent
       location
      :parameters (?dir - direction ?p - player ?
          l1 - location ?l2 - location)
      :precondition (and (at ?p ?l1) (connected ?
          l1 ?dir ?l2) (not (blocked ?l1 ?dir ?
          l2)))
      :effect (and (at ?p ?l2) (not (at ?p ?l1)))
   )

   (:action get ; pick up an item and put it in
       the inventory
      :parameters (?item - item ?p - player ?l1 -
           location)
      :precondition (and (at ?p ?l1) (at ?item ?
          l1))
      :effect (and (inventory ?p ?item) (not (at
          ?item ?l1)))
   )

   (:action get_bamboo_container; get a bamboo
       container using surrounding bamboo
      :parameters (?p - player ?loc - location)
      :precondition (and (at ?p ?loc) (has_bamboo
           ?loc))
      :effect (inventory ?p bamboo_container)
   )

   (:action collect_rain_water
      :parameters (?p - player ?loc - location)
      :precondition (and (at ?p ?loc) (inventory
          ?p bamboo_container) (has_rainfall ?
          loc))
      :effect (and (inventory ?p water) (not (
          treated water)))
   )

   (:action create_fire
      :parameters (?p - player ?loc - location)
      :precondition (and (at ?p ?loc) (inventory
          ?p stone) (inventory ?p wood))
      :effect (and (at fire ?loc) (not (inventory
           ?p stone)) (not (inventory ?p wood)))
   )

   (:action treat_water
      :parameters (?p - player ?loc - location)
      :precondition (and (inventory ?p water) (
          not (treated water)) (at fire ?loc))
      :effect (and (treated water))
   )

   (:action drink_water
      :parameters (?p - player)
      :precondition (and (inventory ?p water) (
          treated water))
      :effect (not (is dehydrated ?p))
   )

   (:action drink_untreated_water
      :parameters (?p - player)
      :precondition (and (inventory ?p water) (
          not (treated water)))
      :effect (is ill ?p)
   )

   (:action create_sos_sign
      :parameters (?p - player)
      :precondition (and (inventory ?p stone) (at
           ?p basecamp))
      :effect (and (not (inventory ?p stone)) (at
           sos_sign basecamp))
```

```
  )

  (:action pick_fruit
     :parameters (?p - player ?loc - location)
     :precondition (and (at ?p ?loc) (has_fruit
         ?loc))
     :effect (inventory ?p fruit)
  )

  (:action eat_fruit
     :parameters (?p - player)
     :precondition (and (is hungry ?p) (
         inventory ?p fruit))
     :effect (and (not (inventory ?p fruit)) (
         not (is hungry ?p)))
  )

  (:action escape
     :parameters (?p - player)
     :precondition (and (at ?p basecamp) (at
         sos_sign basecamp) (not (is hungry ?p))
          (not (is dehydrated ?p)) (not (is ill
          ?p)))
     :effect (not (at ?p basecamp))
  )
)
```

Below is an annotated $\mathbb{PF}$ of the above:

```
(define (problem escape)
  (:domain survive_in_the_jungle)

  (:objects
     npc - player
     jungle bamboo_forrest basecamp - location
     in out north south east west up down -
         direction
     stone wood sos_sign - item
     ill dehydrated hungry - condition
  )

  (:init
     (at npc basecamp)
     (connected basecamp west bamboo_forrest)
     (connected bamboo_forrest east basecamp)
     (connected basecamp east jungle)
     (connected jungle west basecamp)

     (has_bamboo bamboo_forrest)
     (has_fruit jungle)
     (has_rainfall jungle)

     (at stone bamboo_forrest)
     (at stone basecamp)
     (at wood jungle)

     (is dehydrated npc)
     (is hungry npc)
  )

  (:goal (and (not (at npc basecamp)) (not (at
      npc bamboo_forrest)) (not (at npc jungle)
      ) ))
)
```

## C Few-shot Examples

```
Example 1:
action: slide_straw_over_skewer

text: Slide a segment of plastic straw over the
    2 skewers. Cut a segment of a plastic straw
    that is equal to the width between the wheel
     wells on your car. Then, slide it onto 1 of
     the skewers that's attached to a wheel. Do
    the same with the other skewer.

 pddl_action:
 (:action slide_straw_over_skewer
 :parameters (?straw1 - straw ?straw2 - straw ?
     skewer1 - skewer ?skewer2 - skewer ?wheel1
     - wheel ?wheel2 - wheel)
 :precondition (and
 (attached ?skewer1 ?wheel1)
 (attached ?skewer2 ?wheel2)
 (equal_width ?straw1 ?skewer1)
 (equal_width ?straw2 ?skewer2))
 :effect (and
 (over ?straw1 ?skewer1)
 (over ?straw2 ?skewer2)))

entity_state_inference:
- action name: slide_straw_over_skewer
- summarize action: The action involves cutting
    a segment of a plastic straw and sliding it
    over two skewers attached to wheels. The
    straw segments should be equal to the width
    between the wheel wells on a car.
- what is needed to do the action: A segment of
    a plastic straw, two skewers attached to
    wheels.

- what is the result: The straw segments are
    slid over the skewers, which helps in
    maintaining the structure of the car and
    providing stability to the wheels.

- entities:
 - entity name: plastic straw segment
 - before: Not cut and not on skewers
 - after: Cut and slid onto skewers
 - entity name: skewer
 - before: Without straw segments
 - after: With straw segments slid onto them
 - entity name: wheel
 - before: Attached to skewers, without straw
     segments on the skewers
 - after: Attached to skewers, with straw
     segments on the skewers

Example 2:
action: mix_sand_with_color

text: Close the bags tightly, then shake and
    squeeze them to mix the sand. Zip the bags
    shut tightly first. Next, shake the bags to
    help distribute the food coloring. You can
    also gently squeeze and massage the sand
    through the bags. Keep doing this until the
    color is even. Add more food coloring if the
     color is too light for you. Keep in mind
    that the sand will dry lighter.

pddl_action:
(:action mix_sand_with_color
:parameters (?bag - bag ?sand - sand ?color -
    food_coloring)
:precondition (and (sealed ?bag) (contains ?bag
    ?sand) (contains ?bag ?color) (not (
```

```
    evenly_colored ?sand)))
:effect (evenly_colored ?sand))''',

entity_state_inference:
- action name: mix_sand_with_color
- summarize action: This action involves mixing
    colored sand by sealing bags containing sand
     and food coloring, then shaking, squeezing,
     and massaging the bags until the sand color
     is evenly distributed. More food coloring
    may be added if the color is too light, with
     the understanding that the sand will dry
    lighter.
- what is needed to do the action: Sealed bags
    containing sand and food coloring, ability
    to shake and squeeze the bags.
- what is the result: The sand inside the bags
    is evenly colored.

- entities:
 - entity name: bags
 - before: contains uncolored or unevenly
     colored sand and food coloring
 - after: contains evenly colored sand
 - entity name: sand
 - before: uncolored or unevenly colored
 - after: evenly colored
 - entity name: food_coloring
 - before: present in the bags
 - after: mixed evenly with the sand

Example 3:
action: melt_wax

text: Melt candle wax flakes or cubes in a
    double boiler. Set a can, jar, or another
    heat-proof container in a pot, and place
    your solid candle wax (soy or paraffin are
    most common) inside. Surround the container
    with water, filling the pot about halfway up
     the container, then heat the pot on medium
    heat to double boil the wax to melt it
    completely, stirring every minute or so to
    make sure it melts evenly.

pddl_action:
(:action melt_wax
 :parameters (?wax ?container ?pot ?heat_source
     ?water)
 :precondition (and (solid ?wax) (in ?wax ?
     container) (heatproof ?container) (in ?
     container ?pot) (in ?water ?pot) (cold ?
     water))
 :effect (and (liquid ?wax) (heated ?water)))

entity_state_inference:
- action name: melt_wax
- summarize action: This action involves melting
     solid candle wax using a double boiler
    method. The solid wax is placed in a heat-
    proof container, which is then placed in a
    pot filled with water. The pot is heated,
    and the wax is stirred until it melts
    completely.
- what is needed to do the action: The action
    requires solid wax, a heat-proof container,
    a pot, water, and a heat source.
- what is the result: The solid wax is melted
    into liquid wax.

- entities:
 - entity name: wax
 - before: solid
 - after: liquid
 - entity name: container
 - before: empty or containing solid wax
 - after: containing liquid wax
 - entity name: pot
 - before: empty or containing water and
     container with solid wax
 - after: containing water and container with
     liquid wax
 - entity name: water
 - before: cold or room temperature
 - after: heated
 - entity name: heat_source
 - before: off
 - after: on
```

# D Prompts

For reproducibility, we provide the verbatim prompts that we used in the above experiments.

## D.1 Prompt without ZPD

Could you fill out the below PDDL actions with the predicates based on the text?
All fields: parameters, precondition and effect, should have predicates.
For each action, do NOT change the name and do NOT drop the action and do NOT add more actions.
The output should be in correct PDDL format.

<wikiHow text and domain header>

here are the actions I want:
<insert_action_names>
here are the types I have:
<insert_types>
here are the predicates I have:
<insert_predicates>
here are the texts containing steps to <insert_goal>:
<insert_text>

**Example Completion:**

```
(:action clean_water
   :parameters (?player - human ?water - water)
   :precondition (inventory ?player ?water)
   :effect (treated ?water)
)
```

## D.2 Prompt with ZPD

Could you fill out the below PDDL actions with the predicates based on the text? All fields: parameters, precondition and effect, should have

predicates.
For each action, do NOT change the name and do NOT drop the action and do NOT add more actions and:
First, summarize the action in a few sentences based on the text and provide its requirements and its aims if it has.
Next, identify ALL the entities involved in the action and describe whether it changed, unchanged, added, removed in the action in natural language.
Last, translate it into PDDL format. Check all the related entities are in the 'parameters'.

Please use this output format:
- action name: ...
- summarize action: ...
- what is needed to do the action: ...
- what is the result: ...

- entities:
- entity name: ...
- before: ...
- after: ...

- describe how to match it to relevant predicates step by step:
1. ...
2. ...

<wikiHow text and domain header>

here are the actions I want:
<insert_action_names>

here are the types I have:
<insert_types>

here are the predicates I have:
<insert_predicates>

here are the texts containing steps to <insert_goal>:
<insert_text>

**Example Completion:**
- action name: clean_water
- summarize action: The player cleans water in their inventory using heat from a fire.
- what is needed to do the action: The player must have untreated water in their inventory and be at a location with fire.
- what is the result: The player has treated water in their inventory.

- entities:
- entity name: player
- before: Having untreated water in inventory.
- after: Having treated water in inventory.
- entity name: water
- before: Untreated.
- after: Treated.

- describe how to match it to pddl relevant predicates step by step:
1. Check if the player has untreated water in their inventory.
2. Check if the player is at a location with a fire.
3. Replace untreated water with treated water in the player's inventory in the effect.

PDDL:
(:action clean_water
:parameters (?player - human ?loc - location ?water - water)
:precondition (and (at ?player ?loc) (inventory ?player ?water) (not (treated ?water)) (has_fire ?loc))
:effect (treated ?water)
)

## E Calculating Actions Equivalence

The distance between two actions can be divided to two parts:

1. The distance between parameters:

   We do not need to consider the specific parameter names; we only need to consider the parameter types. For each parameter in Action1, we iterate over the parameter list of Action2 to find the first parameter in Action2 with the same type. We use two hash maps, p1 and p2, to record these two parameters and their corresponding types. We increment the counter by 1, remove that parameter from the parameter list of Action2, and break from the current loop. After the iteration, we obtain the number of matching parameters, n. The distance between parameters can be calculated as $|\text{number of parameters in Action1} - n| + |\text{number of parameters in Action2} - n|$.

2. The distance between preconditions/effects:

   For each condition in Action1, we iterate over the condition list of Action2. The conditions can only match if they have the same predicate and

the same number of parameters. We iterate over the parameters in these conditions and make the following judgments:

- If neither of the two current parameters has appeared before (in p1 and p2) and these parameters are not identical, they don't match.
- If the two parameters have different categories, they don't match.
- If the two parameters have the same categories and don't have an index, we consider them as the same parameter entity and give them the same index. We continue the iteration.
- If the two parameters already have indexes, we check if the indexes are equal. If they are not equal, they don't match. Otherwise, we continue the iteration.
- In any other case, they don't match.

If all parameters of the two conditions match, we increment n by 1. The distance between preconditions/effects can be calculated as $|$number of preconditions/effects in Action1 $- n|+|$number of preconditions/effects in Action2$- n|$.