On the Limit of Language Models as Planning Formalizers

Anonymous ACL submission

Abstract

Large Language Models have been shown to fail to create executable and verifiable plans in grounded environments. An emerging line of work shows success in using LLM as a formalizer to generate a formal representation (e.g., PDDL) of the planning domain, which can be deterministically solved to find a plan. We systematically evaluate this methodology while bridging some major gaps. While previous work only generates a partial PDDL representation given templated and thus unrealistic environment descriptions, we generate the complete representation given descriptions of various naturalness levels. Among an array of observations critical to improve LLMs' formal planning ability, we note that large enough models can effectively formalize descriptions as PDDL, outperforming those directly generating plans, while being robust to lexical perturbation. As the descriptions become more natural-sounding, we observe a decrease in performance and provide detailed error analysis.[1]

1 Introduction

Large language models (LLMs) can make *informal plans*, such as suggesting ideas for parties or giving general advice on immigration. However, most users, let alone automated agents like robots, would not be able to actually execute those plans step-by-step to fruition – either to organize parties or acquire visas – without significant prior knowledge or external help. This inability to make executable plans lies in LLMs' inability of grounding and formal reasoning (Liu et al., 2023b; Pan et al., 2023; Zhang et al., 2023). Cutting-edge research in the community has evaluated LLMs' ability to make *formal plans* in grounded environments, such as textual simulations, where all objects and actions represent actualities in the real world. Therefore, any resulting plan that formally involves those objects and actions would be executable and verifiable by nature. Although formal planning has been desirable in the history of AI (Weld, 1999), recent work found that even state-of-the-art LLMs are unable to generate formal plans (Silver et al., 2024; Valmeekam et al., 2024; Stechly et al., 2024).

Instead of using the LLM as a planner to generate the plan directly, an alternative line of work uses the LLM as a *formalizer*. Here, the LLM generates a formal representation of a planning domain, for example in the planning domain definition language (PDDL), based on some natural language descriptions of the environment. This representation can then be fed into a solver to find the plan deterministically (see Figure 1). Previous work achieved great success by showing that LLM-as-formalizer greatly outperforms LLM-as-planner in various domains (Lyu et al., 2023; Xie et al., 2023; Liu et al., 2023a; Zhang et al., 2024a; Zuo et al., 2024; Zhang et al., 2024c; Zhu et al., 2024), as LLMs are more capable of information extraction than formal reasoning (Zhang et al., 2024b). However, the above work has two major shortcomings. First, to simplify the task and evaluation, most have only attempted to generate a partial PDDL representation while assuming the rest is provided, often unrealistic in real life. Second, the language used to describe the environments is often artificially templated and structured, leading to potential overestimation of models' ability.

This paper explores the limit of LLM-as-formalizer devoid of the above two simplifications. We use LLMs to generate the entirety of a PDDL representation, including the domain file and the problem file, given a natural-sounding description of the environment and the task (see Figure 1). On 4 widely used planning simulations from the International Planning Competition, we benchmark a suite of LLMs on generating PDDL that is both solvable and correct. As the descriptions in these

[1] Our code and data can be found at https://anonymous.4open.science/r/llm-as-pddl-formalizer-1BE2.

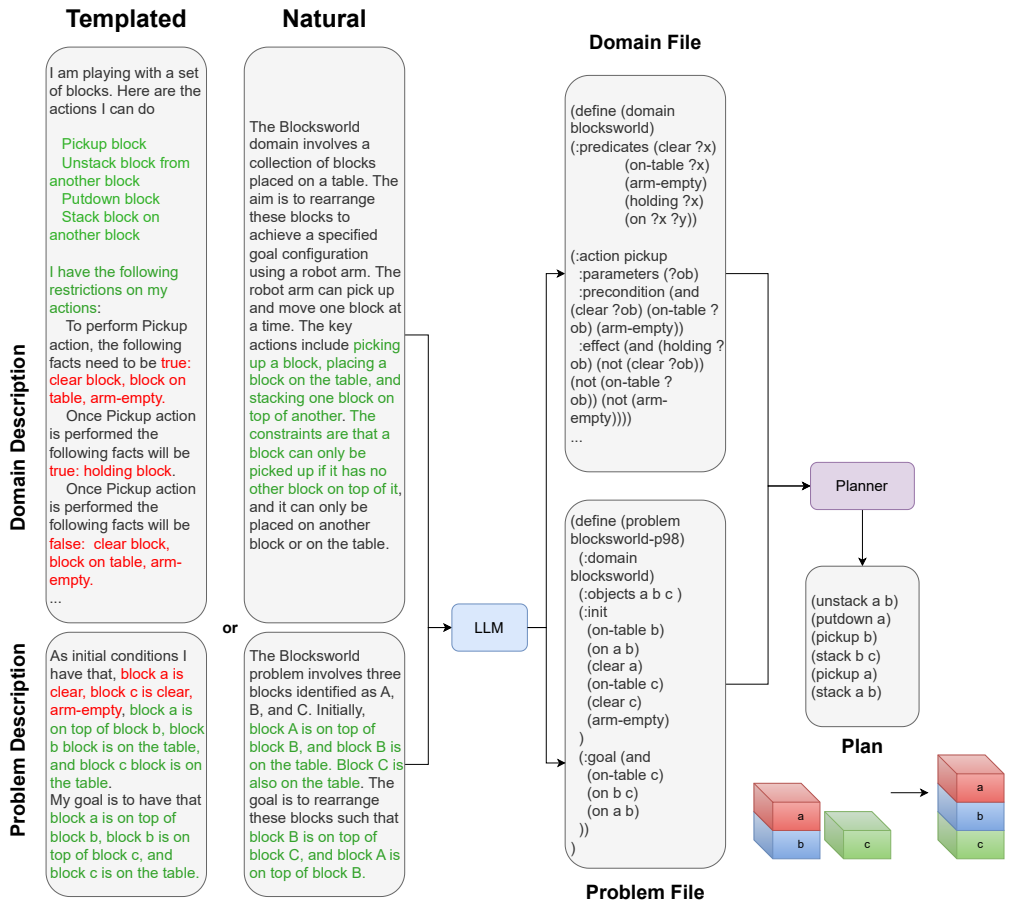

Figure 1: LLM-as-formalizer uses natural language descriptions to generate the Domain and Problem File in PDDL, then these are given to a planner to find a plan. We explore the effect of natural-ness of the language in the description, by giving the model both templated and natural descriptions. Examples of Domain Descriptions and Problem Descriptions from the Blocksworld Domain are shown. The green text displays what the two examples have in common (listing all possible actions and restrictions) and the red text displays text that is not considered natural. The "Templated" text corresponds to the "Heavily Templated" version discussed in Section 4.

datasets are templated, we also construct model-generated, human-validated descriptions that are natural-sounding to different levels.

Our work is the first to systematically analyze state-of-the-art LLMs' ability of the trending methodology of LLM-as-formalizer on the highly challenging task of formal planning. We put forward an array of observations that will benefit future efforts. Discussed in detail in Section 5, our key findings are as follows.

- On various planning simulations, closed-source models like GPT can decently generate entire PDDL, while open-source models like Llama up to 405B cannot.
- When feasible, LLM-as-formalizer greatly outperforms LLM-as-planner.
- As the environment descriptions sound more human-like, the models are more challenged.
- The performance of LLM-as-formalizer is robust to lexical perturbation, while that of LLM-as-planner is not.
- Open-source models succumb to syntax errors unlike GPT models, while semantic errors are common for both types of models.

2 Task: Formal Planning with PDDL

Formal planning (or classical planning) with PDDL involves a domain file ($\mathbb{DF}$) and problem file ($\mathbb{PF}$) (Figure 1). $\mathbb{DF}$ describes general properties in a planning domain that holds true across problems, while $\mathbb{PF}$ describes specific configurations of each problem instance. Concretely, the $\mathbb{DF}$ defines all available actions (and their parameters and pre- and

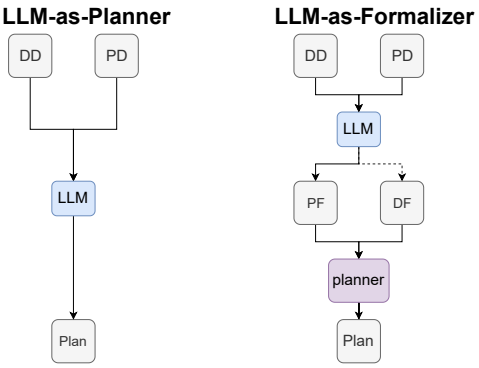

Figure 2: Methodologies for using LLMs in planning. LLM-as-Planner generates the plan directly, while LLM-as-Formalizer translates the $\mathbb{DD}$ and $\mathbb{PD}$ into PDDL. Previous work like Liu et al. (2023a) use the LLM to generate partial PDDL such as $\mathbb{PF}$ only, while we generate the entire PDDL including $\mathbb{PF}$ and $\mathbb{DF}$. Note the $\mathbb{DD}$ and $\mathbb{PD}$ are always provided and the $\mathbb{DF}$ and $\mathbb{PF}$ are always generated by the LLM.

post-conditions) for a state-based environment as well as predicates that represent the properties of object types. The $\mathbb{PF}$ defines the involved objects, the initial states, and the goal states. These two files are then given to a deterministic planner which will algorithmically search for a plan. Such a plan is a series of executable, instantiated actions that sequentially transition the world states from initial to goal. In the AI community, classical planning has been deemed an effective approach to solve real-world users' problems, as the process is precise, explainable, verifiable, and deterministic.

However, formal planning demands a well-crafted pair of $\mathbb{DF}$ and $\mathbb{PF}$. In a real-world planning scenario, an average user would not describe their environment and problem with PDDL, but more likely with a textual description of the domain ($\mathbb{DD}$) and the problem ($\mathbb{PD}$), which can be specific or loose. Hence, we focus on the textual flavor of formal planning: given $\mathbb{DD}$ and $\mathbb{PD}$, the model outputs a successful plan with regard to the $\mathbb{DF}$ and $\mathbb{PF}$ that are withheld from the model.

3 Methodology: LLM-as-Formalizer

To tackle the task above, recent work involving LLMs diverged into two methodologies. The first, **LLM-as-planner**, directly uses LLMs to generate a plan based on the $\mathbb{DD}$ and a $\mathbb{PD}$. The second, **LLM-as-formalizer**, uses LLMs to recover the $\mathbb{DF}$ and $\mathbb{PF}$, before using a deterministic planner to arrive at the plan (Figure 2). Our work will focus on the second while using the first as a baseline. LLM-as-formalizer is in essence neurosymbolic, where LLMs help define the structured representation that is otherwise prohibitively costly to annotate and brittle to generalize. Existing works in this line demonstrated success but only generated a partial PDDL representation, while assuming the rest, including $\mathbb{PF}$ goals (Lyu et al., 2023; Xie et al., 2023), the $\mathbb{PF}$ (Liu et al., 2023a; Zhang et al., 2024a; Zuo et al., 2024), the action semantics in the $\mathbb{DF}$ (Zhang et al., 2024c; Zhu et al., 2024), and the domain file (Wong et al., 2023; Guan et al., 2023). While this simplifies the task and evaluation, the assumption of provided PDDL components is often unrealistic. Therefore, we bridge this gap by asking the LLM to predict the entire PDDL – both the $\mathbb{DF}$ and $\mathbb{PF}$.[2]

4 Evaluation: Datasets, Metrics, Models

To evaluate both approaches above, we work with *fully-observed* textual environments. Here, the provided $\mathbb{DD}$ and $\mathbb{PF}$ contain all necessary information for the model to make a complete plan.

4.1 Datasets

We consider four simulated planning domains, BlocksWorld, Logistics, Barman from the International Planning Competition (IPC, 1998), and MysteryBlocksWorld (Valmeekam et al., 2024).

BlocksWorld, also used in Liu et al. (2023a), is a domain to rearrange stacks of blocks on a table using a robotic arm. There is 1 type of entities, 5 predicates, and 4 actions.

Mystery BlocksWorld obfuscates the original BlocksWorld domain by replacing all the names of the types, predicates, actions, and objects with nonsensical words, akin to a *wug test* (Berko, 1958). This dataset as an control group can effectively test whether models create plans via lexical pattern-matching and memorization.

Logistics, also used in Guan et al. (2023), is a domain to transport packages across different locations using both trucks and airplanes. In this domain, there are 6 types of entities, 3 predicates, and 6 actions.

Barman, also used in Zhu et al. (2024), is a domain to to create cocktails from ingredients using different containers and two robotic arms. In this domain, there are 7 types of entities, 13 predicates, and 12 actions.

[2]It is however minimally necessary to provide the action space, the identifiers and parameters of the actions in $\mathbb{DF}$, so the agent knows *what* actions are possible.

Each dataset comes with ground-truth PDDL describing domains ($\mathbb{DF}$) and problems ($\mathbb{PF}$). The input to the model is a natural language description of the domain ($\mathbb{DD}$) and the problem ($\mathbb{DD}$). The output of the model is a plan, namely a sequence of instantiated actions defined in $\mathbb{DF}$. For each of these datasets, the natural language description $\mathbb{DD}$s and $\mathbb{PD}$s were created in 3 different levels of naturalness.

Heavily Templated. For BlocksWorld, Logistics and Barman, the heavily templated $\mathbb{DD}$ and $\mathbb{PD}$ are generated using the same template as Mystery-Blocksworld (Valmeekam et al., 2024). This description is almost a word-by-word translation of PDDL. For example, for the 'pick-up' action in BlocksWorld, the ground-truth PDDL $\mathbb{DF}$ would be the following:

```
(:action pick-up
:parameters (?b - block)
:precondition (and (clear ?b) (on-table ?b)
(arm-empty))
:effect (and (not (on-table ?b)) (not
(clear ?b)) (not (arm-empty)) (holding ?b))
)
```

while the Heavily Templated $\mathbb{DD}$ is:

```
To perform Pickup action, the following
facts need to be true: clear block, block
on table, arm-empty.
Once Pickup action is performed the
following facts will be true: holding
block.
Once Pickup action is performed the
following facts will be false: clear block,
block on table, arm-empty.
```

From an application point of view, spelling out all preconditions and effects in terms of the predicates is paradoxical, as it assumes the user already have the algorithmic awareness of PDDL.

Moderately Templated. The $\mathbb{DD}$ and $\mathbb{PD}$ are generated using the same template as the original BlocksWorld dataset, following Valmeekam et al. (2024). For example. the Moderately Templated description of the 'pick-up' action is:

```
I can only pick up or unstack one block at
a time.
I can only pick up or unstack a block if my
hand is empty.
I can only pick up a block if the block is
clear. A block is clear if the block has no
other blocks on top of it and if the block
is not picked up.
```

While more natural-sounding than the Heavily Templated version, the description still explicitly discusses the preconditions and effects as well as predicates like 'clear'. Moderately Templated datasets are available only for BlocksWorld and Logistics due to the size of Barman.

Natural. A realistic pair of $\mathbb{DD}$ and $\mathbb{PD}$ should emulate how real-life users would describe the planning domain and problem, such that a human problem solver would understand and have just enough information to find a plan. To create such descriptions, we use a human-in-the-loop, model-assisted data generation approach.

To generate $\mathbb{DD}$, we ask GPT-4o with high temperature to generate and paraphrase a seed annotated $\mathbb{DD}$, and then manually verify the correctness by making sure it lists the correct predicates, preconditions and effects, which are not unique. We next verify the naturalness of the generated text by making there were variations in language throughout all descriptions generated, but was not giving out unnecessary information.

To generate $\mathbb{PD}$, we provide the model with a symbolic configuration that contains the number blocks, the initial stack configuration and the goal stack configuration. The model then 'humanize' the problem by making it sound natural, given a couple of seed exemplars. We manually verify the correctness of the dataset of the non-templated problems by hand by comparing them against the problem configurations. We then verify the naturalness of the $\mathbb{PD}$ by making sure there is variation but no ambiguity in its language.

```
The robot arm can pick up and move one block
at a time from one position to another. It
is only able to move the top block from
any stack or table, and have only one block
held by the robot arm at a time. The main
actions available are 'pick up', ...
```

The above example of the Natural description no longer discusses the preconditions and effects of each actions one by one, but rather focuses on the general rules to the domain. These rules apply to not only 'pick-up' but also other actions. Therefore, the $\mathbb{DD}$ can be much more concise, requires less algorithmic awareness, and more realistic.

In total, we construct 100 problems varying in complexity for all domains. For each of the two Templated descriptions, there is 1 $\mathbb{DD}$ paired with each of 100 $\mathbb{PD}$s. For the Natural description, there are 100 different pairs of $\mathbb{DD}$s and $\mathbb{PD}$s. We refer to these dataset as `BlocksWorld-100`, `MysteryBlocksWorld-100`, `Logistics-100` and `Barman-100`. Data examples can be found in Appendix A.

4.2 Metrics

Following past work (Guan et al., 2023; Zhu et al., 2024), the model-predicted plan is validated us-

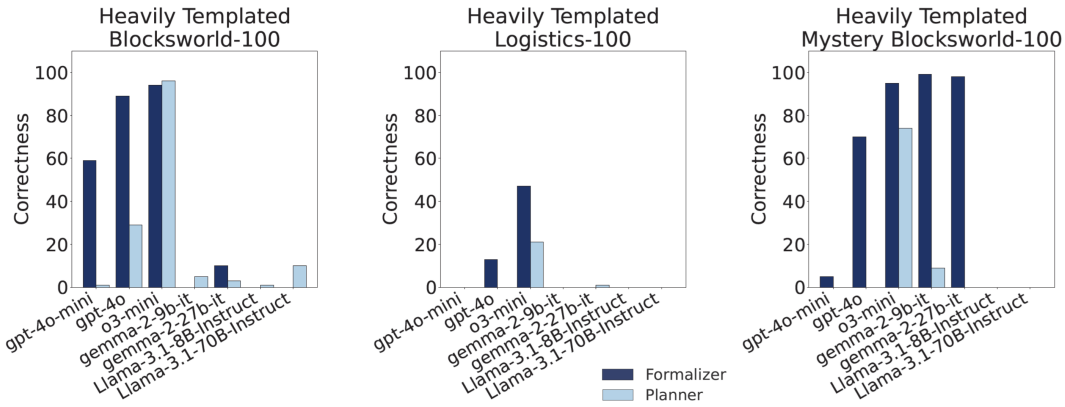

Figure 3: Performance of both usages of LLMs on Heavily Templated `BlocksWorld-100`, `Logistics-100`, and `MysteryBlocksWorld-100`. Detailed results are shown in Appendix C.

ing VAL (Howey et al., 2004) against the ground-truth $\mathbb{DF}$ and $\mathbb{PF}$ provided above, instead of being compared against "ground-truth" plans like some work (Lyu et al., 2023; Liu et al., 2023b; Pan et al., 2023) since there could be multiple correct plans. For the LLM-as-formalizer approach, the predicted $\mathbb{DF}$ and $\mathbb{PF}$ are similarly not compared against the ground-truth, as only the eventual plan is validated because there might be more than one way to formalize the planning domain and problem in PDDL.

We evaluate the predicted plans following Zuo et al. (2024): *solvability* and *correctness*. Solvability only applies to LLM-as-formalizer and indicates the percentage solvable *predicted* $\mathbb{DF}$ and $\mathbb{PF}$, regardless of whether the resulting plan can be executed bsaed on the the *gold* $\mathbb{DF}$ and $\mathbb{PF}$. Correctness indicates the percentage of actually correct plans. Solvability was determined using the planner `dual-bfws-ffparser` implemented by Muise (2016) and Correctness was evaluated using VAL[3].

4.3 Models

For both of the LLM-as-planner and LLM-as-formalizer approach, we consider a number of models, including open-source and closed-source LLMs varying in size, including gemma-2-9b|27b-it (Team et al., 2024), llama-3.1-8B|70B|405B-Instruct (Dubey et al., 2024), DeepSeek-R1-Distill-Llama-8B|70B (Guo et al., 2025)[4], gpt-4o-mini-2024-07-18, gpt-4o-2024-08-06, and o3-mini-2025-01-31[5]. We query these models using KANI (Zhu et al., 2023) with default hyper-parameters. The open-source models are run using 4 RTX A6000 GPUs, averaging about 1062 input and output tokens for the LLM-as-formalizer approach in `BlocksWorld-100`. To emulate real-life application with minimal user interference, we use zero-shot prompts for all naturalness levels across all datasets (see prompts in Appendix B).

[3] `nms.kcl.ac.uk/planning/software/val.html`

[4] Due to the cost and zero solvability on our easiest dataset, results of Llama-405B and DeepSeek-8B|70B are ommitted.

[5] `platform.openai.com/docs/models`

5 Results and Observations

In this section, we display our results as well as perform an in-depth analysis of the strengths and weaknesses of LLMs in formal planning, to understand the impact of the model choice, naturalness of the description, content of the task, and difficulty of the problem.

5.1 Can LLMs formalize?

We seek to understand the extent to which LLMs can act as a formalizer to generate *entire* PDDL, instead of partial components in previous work. Figure 3 displays the results on our experiments using the most natural sounding descriptions on Heavily Templated `BlocksWorld-100`, `Logistics-100` and `MysteryBlocksWorld-100`. Results on the most complex domain `Barman-100` is omitted due to close-to-zero performance for all models.

These results demonstrate that **GPT-family LLMs can decently generate PDDL**, while open-source models even up to 405B parameters struggle. As formalizer, `gpt-4o-mini`, `gpt-4o`, `o3-mini` demonstrate non-trivial and increasing performance on `BlocksWorld-100`. On the more complex `Logistics-100`, `gpt-4o-mini` succumbs to zero performance whereas the other two show decreased performance. The solvability of `gpt-4o-mini` is often much higher than its correctness, suggesting a good grasp of the

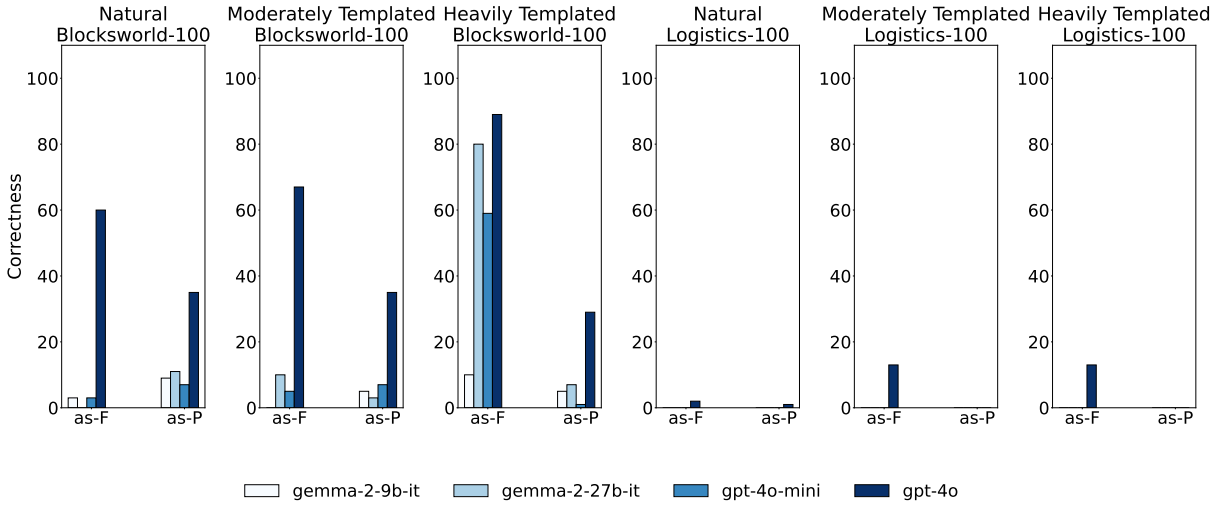

Figure 4: Performance of LLM-as-planner (as-P) and LLM-as-formalzier (as-F) across different naturalness level of description on `BlocksWorld-100` and `Logistics-100`. Detailed results are shown in Appendix C.

PDDL syntax but a lack of semantic understanding. In contrast, the solvability of `gpt-4o` and `o3-mini` is often 80% to 100% of their correctness. On the other hand, **open-sourced models can rarely generate PDDL**, a low-resource language, despite them being reportedly strong at generating high-resource languages like Python (Cassano et al., 2022). All Llama models up to 405B cannot generate any solvable PDDL across all three datasets, while gemma models show poor though non-zero performance on `BlocksWorld-100` and `Logistics-100`, and strong performance on `MysteryBlocksWorld-100`.

5.2 Should LLMs formalize?

Between LLM-as-planner and LLM-as-formalizer, which is the preferred methodology? Figure 3 shows that on `BlocksWorld-100`, `gpt-4o` is able to generate solvable PDDL 64/100 times, and of those 64 plans, 60 of them are correct. This far surpasses the LLM-as-planner baseline, which only found correct plans 33/100 times. This trend holds for `Logistics-100` as well as the Moderately Templated and Natural `BlocksWorld-100` data (Figure 4). On `MysteryBlocksWorld-100`, we can see that LLM-as-formalizer can generate 70/100 correct plans, which far surpassed LLM-as-planner which did not find a single correct plan as the description becomes unorthodox. The superiority of LLM-as-formalizer also extends to `gpt-4o-mini` but not `o3-mini`, who shows strong performance as a planner. These results demonstrate that **LLM-as-formalizer greatly outperforms LLM-as-planner** in most cases, whenever these LLMs can foramlize PDDL *at all*. However, these results also show that models that cannot formalize (e.g., Llama models) can still plan, though with close-to-zero performance.

5.3 The more natural, the harder?

We now examine whether using humanized descriptions makes the problem more difficult. Results from Figure 4 show that on `BlocksWorld-100` as the problem sounds more similar to PDDL and less natural, the performance of all the models improves. Similar results hold for the Logistics domain (see results in Appendix C). This suggests that **a more natural-sounding domain and problem description is much more challenging** than templated, less natural sounding descriptions. One potential explanation is that pattern matching a template back to PDDL is much easier than having to first parse all the predicates and objects from a passage. Another reason is a more natural sounding description may leave out implicit common-sense. For example, the Natural `BlocksWorld-100` does not explicitly specify that a block is ‘clear’, because any human who reads that a block is “on top of a stack” can understand that there is no block on top of it and hence ‘clear’ to be moved. However, models often fail to invoke this knowledge and will leave out the ‘clear’ predicate, leading to unsolvable PDDL or incorrect plans.

5.4 Do LLMs memorize pretraining?

Do LLMs generate plans or formalize PDDL based on what they have memorized in their training data? We determine this by looking at the re-

sults on MysteryBlocksWorld-100, a derivative of BlocksWorld where all names are perturbed and nonsensical. From Figure 3, we can see that LLM-as-planner was not able to find a single correct plan using either gpt-4o-mini or gpt-4o. However, gpt-4o-as-formalizer surpassed this baseline with a Correctness score of 70/100. This suggests that **LLM-as-formalizer is robust to lexical perturbation**, and its success is not due to memorization of the domain which is a part of the pretraining data.

5.5 What kind of errors?

In this section, we discuss the kind of errors in PDDL generation. We perform an error analysis on a random 20 sample subset of problems where a plan was not found, or the found plan was not correct. From there, we categorize the errors by syntax errors in either file, semantic errors in the $\mathbb{DF}$, and in the $\mathbb{PF}$. Of the errors in the $\mathbb{DF}$, we determine finer-grained errors such as incorrect action preconditions and effects, incorrect or missing predicates, and missing or incorrect action parameters. The error analysis for Natural BlocksWorld-100, Logistics-100 and Heavily Templated MysteryBlocksWorld-100 and can be found in Table 1.

For the open-source models, the most common error is syntax errors on BlocksWorld-100 and Logistics-100. For example, models repeatedly use the keyword 'preconditions' instead of 'precondition' which might suggest a lack of grasp of the PDDL syntax. Gemma models like gemma-2-27b make significantly less syntax errors (3 out of 20 on BlocksWorld-100) than Llama models like Llama-3.1-70B (20 out of 20), despite being smaller. Despite the syntax errors, **there are still many semantic errors in the $\mathbb{DF}$ and $\mathbb{PF}$**, which include missing predicates. As shown in Table 3, there is a significant gap between the number of plans that were found, and the number of found that were correct. We find that the most common error made was swapping the parameters in the preconditions of the 'stack' action, leading to incorrect plans. While making much less syntax errors, **GPT models frequently suffer from semantic errors**. Interestingly, the most common error made for gpt-4o come from the $\mathbb{PF}$, which is intuitively easier to generate than the $\mathbb{DF}$. Common errors in the $\mathbb{PF}$ include incorrect predicates in the initial state and goal state. Common errors in the $\mathbb{DF}$ is incorrect effects in an action. For example, in the 'unstack' action, the model does not make the next block 'clear' when the top block has been placed in the hand. For MysteryBlocksWorld-100, there are barely any syntax errors or semantic errors in the $\mathbb{PF}$ but rather the most common errors come from the $\mathbb{DF}$. Since this domain is a result of lexical perturbation, formalizing in PDDL is akin to symbolic information extraction and translation, devoid of much use of commonsense knowledge. Due to the heavily templated descriptions, all the predicates would be listed out in the $\mathbb{PD}$ and the model would just need to match them to PDDL syntax in the $\mathbb{PF}$. While a similar essence, this is more of a challenge for $\mathbb{DF}$ since the clauses of preconditions and effects are more involved. From Table 2, a similar trend between BlocksWorld-100 and MysteryBlocksWorld-100 also suggests that the LLM-as-formalizer methodology is robust to such perturbation.

Models	Syntax Error	$\mathbb{DF}$ Error	$\mathbb{PF}$ Error
	Natural BlocksWorld-100		
gemma-2-9b-it	15/20	20/20	20/20
gemma-2-27b-it	3/20	20/20	14/20
Llama-3.1-8B	20/20	20/20	18/20
Llama-3.1-70B	20/20	20/20	17/20
gpt-4o-mini	2/20	20/20	19/20
gpt-4o	2/20	2/20	18/20
	Natural Logistics-100		
gemma-2-9b-it	7/20	20/20	15/20
gemma-2-27b-it	8/20	20/20	20/20
Llama-3.1-8B	20/20	20/20	20/20
Llama-3.1-70B	20/20	20/20	10/20
gpt-4o-mini	2/20	20/20	19/20
gpt-4o	5/20	20/20	19/20
	MysteryBlocksWorld-100		
gpt-4o-mini	6/20	20/20	1/20
gpt-4o	5/20	16/20	0/20

Table 1: Error analysis of LLM-as-formalizer on various datasets, manually annotated on a 20–example subset.

6 Related Work

Planning with LLMs There has been a large amount of research using LLMs for planning tasks. Some use LLMs for informal planning, also known as script or procedure learning (Zhang et al., 2020; Lyu et al., 2021; Lal et al., 2024). While modern LLMs can make coherent and plausible informal plans, they are ungrounded and so lack executability and verifiability. Work that use LLMs for formal planning in grounded environments generally conclude the inability of such LLMs-as-planners (Sil-

Models	Wrong Precondition	Wrong Effect	Missing Predicate	Missing Action	Missing Parameters
			Natural `BlocksWorld-100`		
gpt-4o-mini	11/20	18/20	19/20	1/20	2/20
gpt-4o	0/20	2/20	0/20	0/20	0/20
			Natural `Logistics-100`		
gpt-4o-mini	20/20	16/20	20/20	5/20	17/20
gpt-4o	17/20	9/20	19/20	1/20	9/20
			Heavily Templated `MysteryBlocksWorld-100`		
gpt-4o-mini	14/20	17/20	17/20	0/20	5/20
gpt-4o	13/20	14/20	0/20	0/20	2/20

Table 2: Analysis of errors found in $\mathbb{DF}$ for Natural `BlocksWorld-100` and Heavily Templated `MysteryBlocksWorld-100` out of 20 randomly sampled instances.

ver et al., 2024; Valmeekam et al., 2024; Stechly et al., 2024). Follow-up work tackles this shortcoming by using the LLM as a heuristic, not just a planner, such as by proposing candidate plans that are iteratively verified (Valmeekam et al., 2023; Kambhampati et al., 2024). While we consider the standard LLM-as-planner as a baseline, our focus is on LLM-as-formalizer, an alternative methodology for the same problem.

LLMs as PDDL formalizer Here, LLMs do not provide plans but rather generate the a PDDL representation of the domain and problem, which is then run through a solver to find the plan. This methodology has proven successful in a number of recent works, where the LLM generates different parts but not all of the PDDL for simplified evaluation. Zuo et al. (2024); Zhang et al. (2024a); Liu et al. (2023a) use the LLM to predict the entire $\mathbb{PF}$, while Xie et al. (2023); Lyu et al. (2023) predict just the goal for the $\mathbb{PF}$. Some predict the $\mathbb{DF}$, such as Zhang et al. (2024c); Zhu et al. (2024) that generate the action semantics of the $\mathbb{DF}$ and Wong et al. (2023) who also predicts the predicates from a candidate list. Closest to our work is Guan et al. (2023) which predicts the $\mathbb{DF}$ as well as the $\mathbb{PF}$ goal. However, our work of holistically generating PDDL shows that coming up with the initial state in the $\mathbb{PF}$ is non-trivial (Section 5.5). Moreover, we vary the level of naturalness of descriptions in addition to the templated ones, which prove to be more challenging and insightful (Section 5.3).

While the above discussions pertain to LLMs generating PDDL, many work on embodied agents outside the NLP community tackle similar problems with different focus (Li et al., 2024).

LLM code generation Our work hinges on modern LLMs' ability to generate code (Chen et al., 2021). In addition to writing or debugging programs (Jiang et al., 2024), LLMs are also used to generate formal, interim representations that are not necessarily PDDL for problem solving. For example, Gao et al. (2023); Lyu et al. (2023); Tang et al. (2024) use the LLM to generate executable Python code for solving symbolic problems. In other work, the generated code may not be executable and is provided to another LLMs to facilitate reasoning (Madaan et al., 2022; Zhang et al., 2023).

A table comparing a couple of these works can be seen in Table 7 in the Appendix (Section E).

7 Conclusion

We explore the limit of state-of-the-art LLMs to be used as a PDDL formalizer for planning with natural language descriptions of different naturalness levels. While the LLM-as-formalizer methodology greatly outperforms the LLM-as-planner baseline in various planning domains, we conclude that with zero-shot prompting, only GPT models are sufficiently capable for the task. Therefore, future work should attempt to equip open-source models with similar ability to democratize the ability of making executable plans. We also find that LLM-as-formalizer is robust to lexical perturbation, demonstrating strong performance in long-tail domains that are underrepresented in pretraining. Our work will inform future efforts of using LLM as a planning formalizer, including experiments on partially-observed environments that require exploration and interaction, more complex environments with a larger action space, and so on.

8 Limitation

A common and valid criticism for using those simulations or text problems for evaluation is that these settings may be too contrived and removed from the reality. Nevertheless, it is likely that LLMs' satisfactory performance on these datasets is a necessary condition to success in real life.

While we only consider zero-shot prompting without any attempt for prompt tuning, it is possible that the models' performance significantly increases otherwise. Therefore, experimental results in all settings may be underestimated. Moreover, advanced prompting techniques such as chain-of-thought, self-refine, and voting can all potentially improve model performance. However, the study of those is out of the scope of this work.

While we advocate for the LLM-as-formalizer methodology over LLM-as-planner, the former's success may be dependent on the task. Highly symbolic tasks which can be relatively easily described, like BlocksWorld, are likely to favor LLM-as-formalizer. However, LLM-as-planner might shine in tasks with a more complex action space requiring common-sense knowledge that is easily accessed by pretraining. Furthermore, while we only consider the most straightforward LLM-as-planner prompting method, more involved methods, like Kambhampati et al. (2024) that combines LLM-as-planner with symbolic validation, will likely lead to a stronger baseline.

Since this work uses only the BlocksWorld, Mystery BlocksWorld, Logistics and Barman domains, it is a small toy example to the usage of LLMs as formalizers and are not representative to problems in the real world, which would be much more challenging. This may pose a risk to users using this code on real world problems.

The datasets we use and we propose are all under the MIT License.

Acknowledgment

We thank Peter Clark for providing invaluable input throughout this work and Andrew Zhu for providing technical support on LLM inference.

References

Jean Berko. 1958. The child's learning of english morphology. *Word*, 14(2-3):150–177.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, et al. 2022. Multipl-e: A scalable and extensible approach to benchmarking neural code generation. *arXiv preprint arXiv:2208.08227*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

R. Howey, D. Long, and M. Fox. 2004. Val: automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 294–301.

IPC. 1998. International planning competition. https://www.icaps-conference.org/competitions.

IPC. 2000. International planning competition. https://www.icaps-conference.org/competitions.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. 2024. Llms can't plan, but can help planning in llm-modulo frameworks. *arXiv preprint arXiv:2402.01817*.

Yash Kumar Lal, Li Zhang, Faeze Brahman, Bodhisattwa Prasad Majumder, Peter Clark, and Niket Tandon. 2024. Tailoring with targeted precision: Edit-based agents for open-domain procedure customization. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 15597–15611, Bangkok, Thailand. Association for Computational Linguistics.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li Erran Li, Ruohan Zhang, et al. 2024. Embodied agent interface: Benchmarking llms for embodied decision making. *arXiv preprint arXiv:2410.07166*.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023a. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Hanmeng Liu, Ruoxi Ning, Zhiyang Teng, Jian Liu, Qiji Zhou, and Yue Zhang. 2023b. Evaluating the logical reasoning ability of chatgpt and gpt-4. *arXiv preprint arXiv:2304.03439*.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 305–329, Nusa Dua, Bali. Association for Computational Linguistics.

Qing Lyu, Li Zhang, and Chris Callison-Burch. 2021. Goal-oriented script construction. In *Proceedings of the 14th International Conference on Natural Language Generation*, pages 184–200, Aberdeen, Scotland, UK. Association for Computational Linguistics.

Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. 2022. Language models of code are few-shot commonsense learners. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1384–1403, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Christian Muise. 2016. Planning.Domains. In *The 26th International Conference on Automated Planning and Scheduling - Demonstrations*.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. 2023. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3806–3824, Singapore. Association for Computational Linguistics.

Jendrik Seipp, Álvaro Torralba, and Jörg Hoffmann. 2022. PDDL generators. https://doi.org/10.5281/zenodo.6382173.

Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B Tenenbaum, Leslie Kaelbling, and Michael Katz. 2024. Generalized planning in pddl domains with pretrained large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20256–20264.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. 2024. Chain of thoughtlessness: An analysis of cot in planning. *arXiv preprint arXiv:2405.04776*.

Hao Tang, Darren Key, and Kevin Ellis. 2024. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment. *Preprint*, arXiv:2402.12275.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

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, Sarath Sreedharan, Matthew Marquez, Alberto Olmo, and Subbarao Kambhampati. 2023. On the planning abilities of large language models (a critical investigation with a proposed benchmark). *Preprint*, arXiv:2302.06706.

Daniel S Weld. 1999. Recent advances in ai planning. *AI magazine*, 20(2):93–93.

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. *arXiv preprint arXiv:2302.05128*.

Hongming Zhang, Muhao Chen, Haoyu Wang, Yangqiu Song, and Dan Roth. 2020. Analogous process structure induction for sub-event sequence prediction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1541–1550, Online. Association for Computational Linguistics.

Li Zhang, Peter Jansen, Tianyi Zhang, Peter Clark, Chris Callison-Burch, and Niket Tandon. 2024a. PDDLEGO: Iterative planning in textual environments. In *Proceedings of the 13th Joint Conference on Lexical and Computational Semantics (*SEM 2024)*, pages 212–221, Mexico City, Mexico. Association for Computational Linguistics.

Li Zhang, Hainiu Xu, Abhinav Kommula, Chris Callison-Burch, and Niket Tandon. 2024b. OpenPI2.0: An improved dataset for entity tracking in texts. In *Proceedings of the 18th Conference of the European Chapter of the Association for*

Computational Linguistics (Volume 1: Long Papers), pages 166–178, St. Julian's, Malta. Association for Computational Linguistics.

Li Zhang, Hainiu Xu, Yue Yang, Shuyan Zhou, Weiqiu You, Manni Arora, and Chris Callison-Burch. 2023. Causal reasoning of entities and events in procedural texts. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 415–431, Dubrovnik, Croatia. Association for Computational Linguistics.

Tianyi Zhang, Li Zhang, Zhaoyi Hou, Ziyu Wang, Yuling Gu, Peter Clark, Chris Callison-Burch, and Niket Tandon. 2024c. PROC2PDDL: Open-domain planning representations from texts. In *Proceedings of the 2nd Workshop on Natural Language Reasoning and Structured Explanations (@ACL 2024)*, pages 13–24, Bangkok, Thailand. Association for Computational Linguistics.

Andrew Zhu, Liam Dugan, Alyssa Hwang, and Chris Callison-Burch. 2023. Kani: A lightweight and highly hackable framework for building language model applications. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pages 65–77, Singapore. Association for Computational Linguistics.

Wang Zhu, Ishika Singh, Robin Jia, and Jesse Thomason. 2024. Language models can infer action semantics for classical planners from environment feedback. *arXiv preprint arXiv:2406.02791*.

Max Zuo, Francisco Piedrahita Velez, Xiaochen Li, Michael L Littman, and Stephen H Bach. 2024. Planetarium: A rigorous benchmark for translating text to structured planning languages. *arXiv preprint arXiv:2407.03321*.

A Data Examples

As discussed above, each dataset comes with ground-truth PDDL describing domains ($\mathbb{DF}$) and problems ($\mathbb{PF}$). To maximize flexibility when performing analysis, we construct problem instances ourselves for some datasets, so that we can measure complexity with metrics like the number of Blocks in BlocksWorld, for which we ensure a uniform distribution to avoid biases. Instances for BlocksWorld were randomly generated by varying the number of blocks and number of stacks in the initial and goal states from 2 to 15. Instances for Mystery BlocksWorld were randomly sampled from (Valmeekam et al., 2024). Instances of Logistics were taken directly from (IPC, 1998) and (IPC, 2000). Instances of Barman were generated by using (Seipp et al., 2022) and varying the number of shot-glasses, ingredients and cocktails from 1 to 9. Below are the example PDDL and descriptions for all datasets.

A.1 `BlocksWorld-100` PDDL

The following are an example of the ground-truth $\mathbb{DF}$ and $\mathbb{PF}$ for `BlocksWorld-100`.

$\mathbb{DF}$:

```
(define (domain blocksworld)
(:predicates (clear ?x)
(on-table ?x)
(arm-empty)
(holding ?x)
(on ?x ?y))

(:action pickup
:parameters (?ob)
:precondition (and (clear ?ob) (on-table
?ob) (arm-empty))
:effect (and (holding ?ob) (not (clear
?ob)) (not (on-table ?ob))
(not (arm-empty))))

(:action putdown
:parameters (?ob)
:precondition (holding ?ob)
:effect (and (clear ?ob) (arm-empty)
(on-table ?ob)
(not (holding ?ob))))

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

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

The $\mathbb{DF}$ contains all four actions (pickup, putdown, stack and unstack) and their pre-conditions and post-conditions, as well as predicates needed for the domain.

$\mathbb{PF}$:

```
(define (problem blocksworld-p99)
(:domain blocksworld)
(:objects red blue green yellow )
(:init
(on-table red)
(on blue red)
(clear blue)
(on-table green)
(on yellow green)
(clear yellow)
(arm-empty)
)
(:goal (and
(on-table red)
(on green red)
(on yellow green)
(on blue yellow)
))
)
```

The $\mathbb{PF}$ contains the objects, initial state and goal state for the problem.

A.2 `BlocksWorld-100` $\mathbb{DD}$ and $\mathbb{PD}$

The following display example $\mathbb{DD}$ and $\mathbb{PD}$ for all natural settings in the `BlocksWorld-100` dataset. We can see that the descriptions have all the same components as the $\mathbb{DF}$ and $\mathbb{PF}$ in PDDL, but written in different levels of naturalness.

For the Heavily Templated $\mathbb{DD}$, all preconditions and post-conditions are written out explicitly and sound similar to PDDL. The $\mathbb{PD}$ is similar, in that it lists all the predicates needed for to solve the task.

Heavily Templated $\mathbb{DD}$:

```
I am playing with a set of blocks. Here
are the actions I can do

Pickup block
Unstack block from another block
Putdown block
Stack block on another block

I have the following restrictions on
my actions:
To perform Pickup action, the following
facts need to be true: clear block, block
on table, arm-empty.
Once Pickup action is performed the
following facts will be true: holding
block.
Once Pickup action is performed the
following facts will be false: clear block,
block on table, arm-empty.
To perform Putdown action, the following
facts need to be true: holding block.
Once Putdown action is performed the
following facts will be true: clear block,
block on table, arm-empty.
Once Putdown action is performed the
following facts will be false: holding
block.
To perform Stack action, the following
needs to be true: clear block2, holding
block1.
Once Stack action is performed the
following will be true: arm-empty, clear
block1, block1 on block2.
Once Stack action is performed the
following will be false: clear block2,
holding block1.
To perform Unstack action, the following
needs to be true: block1 on block2, clear
block1, arm-empty.
Once Unstack action is performed the
following will be true: holding block1,
clear block2.
Once Unstack action is performed the
following will be false:, block1 on block2,
clear block1, arm-empty.
```

Heavily Templated $\mathbb{PD}$:

```
As initial conditions I have that, the
blue block is clear, the yellow block is
clear, arm-empty, the blue block is on top
of the red block, the yellow block is on
top of the green block, the red block is
on the table, and the green block is on
the table.
My goal is to have that the blue block is
on top of the yellow block, the green block
is on top of the red block, the yellow
block is on top of the green block, and
the red block is on the table.
```

For the Moderately Templated data, the $\mathbb{DD}$ and $\mathbb{PD}$ are much more natural than the Heavily Templated data, but all predicates are still listed.

Moderately Templated $\mathbb{DD}$:

```
I am playing with a set of blocks where
I need to arrange the blocks into stacks.
Here are the actions I can do

Pick up a block
Unstack a block from on top of another
block
Put down a block
Stack a block on top of another block

I have the following restrictions on
my actions:
I can only pick up or unstack one block at
a time.
I can only pick up or unstack a block if
my hand is empty.
I can only pick up a block if the block
is on the table and the block is clear. A
block is clear if the block has no other
blocks on top of it and if the block is
not picked up.
I can only unstack a block from on top of
another block if the block I am unstacking
was really on top of the other block.
I can only unstack a block from on top of
another block if the block I am unstacking
is clear.
Once I pick up or unstack a block, I am
holding the block.
I can only put down a block that I am
holding.
I can only stack a block on top of another
block if I am holding the block being
stacked.
I can only stack a block on top of another
block if the block onto which I am stacking
the block is clear.
Once I put down or stack a block, my hand
becomes empty.
Once you stack a block on top of a second
block, the second block is no longer clear.
```

Moderately Templated $\mathbb{PD}$:

```
As initial conditions I have that, the blue
block is clear, the yellow block is clear,
the hand is empty, the blue block is on
top of the red block, the yellow block is
on top of the green block, the red block
is on the table, and the green block is on
the table.
My goal is to have that the blue block is
on top of the yellow block, the green block
is on top of the red block, the yellow
block is on top of the green block, and
the red block is on the table.
```

Finally for the natural data, we can see that the $\mathbb{DD}$ and $\mathbb{PD}$ give all necessary information to complete the task, but does not sound like PDDL, and does not describe all predicates needed to perform the task.

Natural $\mathbb{DD}$:

```
The Blocksworld game involves a set of
blocks of different colors, which can be
stacked on top of each other or placed on
the table. The objective is to move the
blocks from an initial configuration to a
goal configuration using a series of legal
moves. Legal moves in Blocksworld include:
picking up a block from the table or from
the top of another block, stacking a block
onto the table, or stacking a block onto
another block.
```

Natural $\mathbb{PD}$:

```
In this particular game, there are 4
blocks: a red block, a blue block, a green
block, and a yellow block. At the start,
the red block is on the table, the blue
block is on top of the red block, the green
block is on the table, and the yellow block
is on top of the green block. The goal is
to have the red block on the table, the
green block on top of the red block, the
yellow block on top of the green block, and
the blue block on top of the yellow block.
```

A.3 `MysteryBlocksWorld-100` PDDL

This section displays an example of the groundtruth $\mathbb{DF}$ and $\mathbb{PF}$ for `MysteryBlocksWorld-100`.

$\mathbb{DF}$:

```
(define (domain mystery_blocksworld)
(:predicates (province ?x)
(planet ?x)
(harmony)
(pain ?x)
(craves ?x ?y))

(:action attack
:parameters (?ob)
:precondition (and (province ?ob) (planet
?ob) (harmony))
:effect (and (pain ?ob) (not (province
?ob)) (not (planet ?ob))
(not (harmony))))

(:action succumb
:parameters (?ob)
:precondition (pain ?ob)
:effect (and (province ?ob) (harmony)
(planet ?ob)
(not (pain ?ob))))

(:action overcome
:parameters (?ob ?underob)
:precondition (and (province ?underob)
(pain ?ob))
:effect (and (harmony) (province ?ob)
(craves ?ob ?underob)
(not (province ?underob)) (not (pain
?ob))))

(:action feast
:parameters (?ob ?underob)
:precondition (and (craves ?ob ?underob)
(province ?ob) (harmony))
:effect (and (pain ?ob) (province ?underob)
(not (craves ?ob ?underob)) (not (province
?ob)) (not (harmony)))))
```

$\mathbb{PF}$:

```
(define (problem mystery_blocksworld-p01)
(:domain mystery_blocksworld)
(:objects a b c d )
(:init
(craves a b)
(craves b c)
(harmony)
(planet c)
(planet d)
(province a)
(province d)
)
(:goal (and
(craves a d)
(craves c a)
))
)
```

A.4 `MysteryBlocksWorld-100` $\mathbb{DD}$ and $\mathbb{PD}$

The following are example $\mathbb{DD}$ and $\mathbb{PD}$ of the Heavily Templated `MysteryBlocksWorld-100`. Text written in green demonstrates natural sounding text while text written in red demonstrates text that sounds the most like PDDL.

$\mathbb{DD}$:

```
I am playing with a set of objects. Here
are the actions I can do

Attack object
Feast object from another object
Succumb object
Overcome object from another object

I have the following restrictions on
my actions:
To perform Attack action, the following
facts need to be true: Province object,
Planet object, Harmony.
Once Attack action is performed the
following facts will be true: Pain object.
Once Attack action is performed the
following facts will be false: Province
object, Planet object, Harmony.
To perform Succumb action, the following
facts need to be true: Pain object.
Once Succumb action is performed the
following facts will be true: Province
object, Planet object, Harmony.
Once Succumb action is performed the
following facts will be false: Pain
object.
To perform Overcome action, the following
needs to be true: Province other object,
Pain object.
Once Overcome action is performed the
following will be true: Harmony, Province
object, Object Craves other object.
Once Overcome action is performed the
following will be false: Province other
object, Pain object.
To perform Feast action, the following
needs to be true: Object Craves other
object, Province object, Harmony.
Once Feast action is performed the
following will be true: Pain object,
Province other object.
Once Feast action is performed the
following will be false:, Object Craves
other object, Province object, Harmony.
```

$\mathbb{PD}$:

```
As initial conditions I have that, object
a craves object b, object b craves object
c, harmony, planet object c, planet object
d, province object a and province object
d.
My goal is to have that object a craves
object d and object c craves object a.
```

A.5 `Logistics-100` PDDL

The following are an example of the groundtruth $\mathbb{DF}$ and $\mathbb{PF}$ for `Logistics-100`.

$\mathbb{DF}$:

```
(define (domain logistics)
(:requirements :strips)
(:predicates (package ?obj)
(truck ?truck)
(airplane ?airplane)
(airport ?airport)
(location ?loc)
(in-city ?obj ?city)
(city ?city)
(at ?obj ?loc)
(in ?obj ?obj))

(:action load-truck
:parameters
(?obj
?truck
?loc)
:precondition
(and (package ?obj) (truck ?truck)
(location ?loc)
(at ?truck ?loc) (at ?obj ?loc))
:effect
(and (not (at ?obj ?loc)) (in ?obj ?truck)))

(:action load-airplane
:parameters
(?obj
?airplane
?loc)
:precondition
(and (package ?obj) (airplane ?airplane)
(location ?loc)
(at ?obj ?loc) (at ?airplane ?loc))
:effect
(and (not (at ?obj ?loc)) (in ?obj
?airplane)))

(:action unload-truck
:parameters
(?obj
?truck
?loc)
:precondition
(and (package ?obj) (truck ?truck)
(location ?loc)
(at ?truck ?loc) (in ?obj ?truck))
:effect
(and (not (in ?obj ?truck)) (at ?obj
?loc)))
```

$\mathbb{DF}$ (cont'd)

```
(:action unload-airplane
:parameters
(?obj
?airplane
?loc)
:precondition
(and (package ?obj) (airplane ?airplane)
(location ?loc)
(in ?obj ?airplane) (at ?airplane ?loc))
:effect
(and (not (in ?obj ?airplane)) (at ?obj
?loc)))

(:action drive-truck
:parameters
(?truck
?loc-from
?loc-to
?city)
:precondition
(and (truck ?truck) (location ?loc-from)
(location ?loc-to) (city ?city)
(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)))

(:action fly-airplane
:parameters
(?airplane
?loc-from
?loc-to)
:precondition
(and (airplane ?airplane) (airport
?loc-from) (airport ?loc-to)
(at ?airplane ?loc-from))
:effect
(and (not (at ?airplane ?loc-from)) (at
?airplane ?loc-to)))
)
```

The $\mathbb{DF}$ contains all six actions (load-truck, load-airplane, unload-truck, unload-airplane, drive-truck, drive-airplane) and their pre-conditions and post-conditions, as well as predicates needed for the domain.

$\mathbb{PF}$:

```
(define (problem logistics-4-0)
(:domain logistics)
(:objects apn1 apt2 pos2 apt1 pos1 cit2
cit1 tru2 tru1 obj23 obj22 obj21 obj13
obj12 obj11 )
(:init (package obj11) (package obj12)
(package obj13)
(package obj21)
(package obj22) (package obj23) (truck
tru1) (truck tru2)
(city cit1) (city cit2)
(location pos1) (location apt1) (location
pos2) (location apt2) (airport apt1)
(airport apt2) (airplane apn1) (at apn1
apt2) (at tru1 pos1) (at obj11 pos1)
(at obj12 pos1) (at obj13 pos1) (at tru2
pos2) (at obj21 pos2) (at obj22 pos2)
(at obj23 pos2) (in-city pos1 cit1)
(in-city apt1 cit1) (in-city pos2 cit2)
(in-city apt2 cit2))
(:goal (and (at obj11 apt1) (at obj23 pos1)
(at obj13 apt1) (at obj21 pos1)))
)
```

The $\mathbb{PF}$ contains the objects, initial state and goal state for the problem.

A.6 `Logistics-100` $\mathbb{DD}$ and $\mathbb{PD}$

The following display example $\mathbb{DD}$ and $\mathbb{PD}$ for all natural settings in the `Logistics-100` dataset. We can see that the descriptions have all the same components as the $\mathbb{DF}$ and $\mathbb{PF}$ in PDDL, but written in different levels of naturalness.

For the Heavily Templated $\mathbb{DD}$, all preconditions and post-conditions are written out explicitly and sound similar to PDDL. The $\mathbb{PD}$ is similar, in that it lists all the predicates needed for to solve the task.

Heavily Templated $\mathbb{DD}$:

```
I need to move packages between locations.
Here are the actions I can do

Load an package onto a truck at a location
(load-truck package truck location)
Load an package onto an airplane at a
location (load-airplane package airplane
location)
Unload an package from a truck at a
location (unload-truck package truck
location)
Unload an package from an airplane at a
location (unload-airplane package airplane
location)
Drive a truck from location1 to location2
in a city (drive-truck truck location1
location2 city)
Fly an airplane from airport1 to airport2
(fly-airplane airplane airport1 airport2)

I have the following restrictions on
my actions:
To perform load-truck action, the following
facts need to be true: o is an package,
t is a truck, l is a location, the truck
is at the location, the package is at the
location.
Once load-truck action is performed the
following facts will be true: the package
is in the truck.
Once load-truck action is performed the
following facts will be false: the package
is at the location.

To perform load-airplane action, the
following facts need to be true: o is
an package, a is an airplane, l is a
location, the airplane is at the location,
the package is at the location.
Once load-airplane action is performed the
following facts will be true: the package
is in the airplane.
Once load-airplane action is performed the
following facts will be false: the package
is at the location.

To perform unload-truck action, the
following facts need to be true: o is an
package, t is a truck, l is a location,
the truck is at the location, the package
is in the truck.
Once unload-truck action is performed the
following facts will be true: the package
is at the location.
Once unload-truck action is performed the
following facts will be false: the package
is in the truck.
```

$\mathbb{DD}$ (cont'd)

```
To perform unload-airplane action, the following facts need to be true: o is an package, a is a airplane, l is a location, the airplane is at the location, the package is in the airplane.
Once unload-airplane action is performed the following facts will be true: the package is at the location.
Once unload-airplane action is performed the following facts will be false: the package is in the airplane.

To perform drive-truck action, the following need to be true: t is a truck, l1 is a location, l2 is a location, c is a city, the truck is at l1, l1 is in the city, l2 is in the city.
Once drive-truck action is performed the following facts will be true: the truck is at l2.
Once drive-truck action is performed the following facts will be false: the truck is at l1.

To perform fly-airplane action, the following must be true: p is an airplane, a1 is an airport, a2 is an airport, the airplane is at a1.
Once fly-airplane is performed the following facts will be true: the airplane is at a2.
Once fly-airplane is performed the following facts will be false: the airplane is at a1.
```

Heavily Templated $\mathbb{PD}$:

```
As initial conditions, I have that, obj11 is a package, obj12 is a package, obj13 is a package, obj21 is a package, obj22 is a package, obj23 is a package, tru1 is a truck, tru2 is a truck, cit1 is a city, cit2 is a city, pos1 is a location, apt1 is a location, pos2 is a location, apt2 is a location, apt1 is an airport, apt2 is an airport, apn1 is an airplane, apn1 is at apt2, tru1 is at pos1, obj11 is at pos1, obj12 is at pos1, obj13 is at pos1, tru2 is at pos2, obj21 is at pos2, obj22 is at pos2, obj23 is at pos2, pos1 is in cit1, apt1 is in cit1, pos2 is in cit2, and apt2 is in cit2.
My goal is to have that obj11 is at apt1, obj23 is at pos1, obj13 is at apt1, and obj21 is at pos1.
```

For the Moderately Templated data, the $\mathbb{DD}$ and $\mathbb{PD}$ are much more natural than the Heavily Templated data, but all predicates are still listed.

Moderately Templated $\mathbb{DD}$:

```
I need to move packages between locations. Here are the actions I can do

Load an package onto a truck at a location (load-truck package truck location)
Load an package onto an airplane at a location (load-airplane package airplane location)
Unload an package from a truck at a location (unload-truck package truck location)
Unload an package from an airplane at a location (unload-airplane package airplane location)
Drive a truck from location1 to location2 in a city (drive-truck truck location1 location2 city)
Fly an airplane from airport1 to airport2 (fly-airplane airplane airport1 airport2)

I have the following restrictions on my actions:
I can only load a package onto a truck or airplane if both the package and airplane are at the location.
Once I load the package in the truck or airplane, it is no longer at the location.
I can only unload a package from a truck or airplane if the truck or airplane is at the location and the package is in the truck or airplane.
Once I unload the truck or airplane, the object is at the location and no longer in the truck or airplane.
I can only drive a truck between locations if the truck is at the first location and both the first and second locations are in the same city. Once I drive a truck, the truck is in the second city and no longer in the first city.
I can only fly an airplane between two airports and the airplane is at the first airport.
Once I fly an airplane, the airplane is at the second airport and no longer at the first airport.
```

The $\mathbb{PD}$ for the Moderately Templated data is the same as the $\mathbb{PD}$ for Heavily Templated data.

Finally for the natural data, we can see that the $\mathbb{DD}$ and $\mathbb{PD}$ give all necessary information to complete the task, but does not sound like PDDL, and does not describe all predicates needed to perform the task.

Natural $\mathbb{DD}$:

```
In the Logistics game, your goal is to transport packages between different locations using trucks and airplanes. Here are the actions you can perform: You can load a package onto a truck at a particular location if both the package and the truck are present there. Similarly, loading a package onto an airplane requires both the package and the airplane to be at the same location. Once a package is loaded onto a truck or airplane, it leaves its original location. To unload a package, the truck or airplane must be at the same location where you want to unload, and the package should be inside the vehicle. When you unload, the package arrives at the new location and exits the vehicle. If you want to drive a truck from one location to another within a city, the truck needs to begin its journey at the starting point, and both locations must be within the city boundaries. After driving, the truck will find itself at the destination, leaving the starting point behind. As for flying, an airplane can travel between two airports, but it must be ready for takeoff from the initial airport. After the flight, the airplane lands at the destination airport, departing from the origin airport in the process.
```

Natural $\mathbb{PD}$:

```
In this logistics scenario, we begin with several objects and locations. Package obj11, obj12, and obj13 are initially at location pos1. Similarly, package obj21, obj22, and obj23 start at location pos2. We have two trucks: truck tru1 is stationed at pos1, and truck tru2 is at pos2. Additionally, we have a single airplane, apn1, which is located at airport apt2. Our map comprises two cities: cit1 and cit2. City cit1 contains location pos1 and airport apt1, while city cit2 includes location pos2 and airport apt2. The ultimate objective is to relocate package obj11 and obj13 to airport apt1 and to move package obj21 and obj23 to location pos1.
```

A.7 `Barman-100` PDDL

The following are an example of the groundtruth $\mathbb{DF}$ and $\mathbb{PF}$ for `Logistics-100`.

$\mathbb{DF}$:

```
(define (domain barman)
(:requirements :strips :typing)
(:types hand level beverage dispenser
container - object
ingredient cocktail - beverage
shot shaker - container)
(:predicates (ontable ?c - container)
(holding ?h - hand ?c - container)
(handempty ?h - hand)
(empty ?c - container)
(contains ?c - container ?b - beverage)
(clean ?c - container)
(used ?c - container ?b - beverage)
(dispenses ?d - dispenser ?i - ingredient)
(shaker-empty-level ?s - shaker ?l - level)
(shaker-level ?s - shaker ?l - level)
(next ?l1 ?l2 - level)
(unshaked ?s - shaker)
(shaked ?s - shaker)
(cocktail-part1 ?c - cocktail ?i -
ingredient)
(cocktail-part2 ?c - cocktail ?i -
ingredient))

(:action grasp
:parameters (?h - hand ?c - container)
:precondition (and (ontable ?c) (handempty
?h))
:effect (and (not (ontable ?c))
(not (handempty ?h))
(holding ?h ?c)))

(:action leave
:parameters (?h - hand ?c - container)
:precondition (holding ?h ?c)
:effect (and (not (holding ?h ?c))
(handempty ?h)
(ontable ?c)))

(:action fill-shot
:parameters (?s - shot ?i - ingredient ?h1
?h2 - hand ?d - dispenser)
:precondition (and (holding ?h1 ?s)
(handempty ?h2)
(dispenses ?d ?i)
(empty ?s)
(clean ?s))
:effect (and (not (empty ?s))
(contains ?s ?i)
(not (clean ?s))
(used ?s ?i)))
```

$\mathbb{DF}$ (cont'd)

```
(:action refill-shot
:parameters (?s - shot ?i - ingredient ?h1
?h2 - hand ?d - dispenser)
:precondition (and (holding ?h1 ?s)
(handempty ?h2)
(dispenses ?d ?i)
(empty ?s)
(used ?s ?i))
:effect (and (not (empty ?s))
(contains ?s ?i)))

(:action empty-shot
:parameters (?h - hand ?p - shot ?b -
beverage)
:precondition (and (holding ?h ?p)
(contains ?p ?b))
:effect (and (not (contains ?p ?b))
(empty ?p)))

(:action clean-shot
:parameters (?s - shot ?b - beverage ?h1
?h2 - hand)
:precondition (and (holding ?h1 ?s)
(handempty ?h2)
(empty ?s)
(used ?s ?b))
:effect (and (not (used ?s ?b))
(clean ?s)))


(:action pour-shot-to-clean-shaker
:parameters (?s - shot ?i - ingredient ?d
- shaker ?h1 - hand ?l ?l1 - level)
:precondition (and (holding ?h1 ?s)
(contains ?s ?i)
(empty ?d)
(clean ?d)
(shaker-level ?d ?l)
(next ?l ?l1))
:effect (and (not (contains ?s ?i))
(empty ?s)
(contains ?d ?i)
(not (empty ?d))
(not (clean ?d))
(unshaked ?d)
(not (shaker-level ?d ?l))
(shaker-level ?d ?l1)))

(:action pour-shot-to-used-shaker
:parameters (?s - shot ?i - ingredient ?d
- shaker ?h1 - hand ?l ?l1 - level)
:precondition (and (holding ?h1 ?s)
(contains ?s ?i)
(unshaked ?d)
(shaker-level ?d ?l)
(next ?l ?l1))
:effect (and (not (contains ?s ?i))
(contains ?d ?i)
(empty ?s)
(not (shaker-level ?d ?l))
(shaker-level ?d ?l1)))
```

$\mathbb{DF}$ (cont'd)

```
(:action empty-shaker
:parameters (?h - hand ?s - shaker ?b -
cocktail ?l ?l1 - level)
:precondition (and (holding ?h ?s)
(contains ?s ?b)
(shaked ?s)
(shaker-level ?s ?l)
(shaker-empty-level ?s ?l1))
:effect (and (not (shaked ?s))
(not (shaker-level ?s ?l))
(shaker-level ?s ?l1)
(not (contains ?s ?b))
(empty ?s)))

(:action clean-shaker
:parameters (?h1 ?h2 - hand ?s - shaker)
:precondition (and (holding ?h1 ?s)
(handempty ?h2)
(empty ?s))
:effect (and (clean ?s)))

(:action shake
:parameters (?b - cocktail ?d1 ?d2 -
ingredient ?s - shaker ?h1 ?h2 - hand)
:precondition (and (holding ?h1 ?s)
(handempty ?h2)
(contains ?s ?d1)
(contains ?s ?d2)
(cocktail-part1 ?b ?d1)
(cocktail-part2 ?b ?d2)
(unshaked ?s))
:effect (and (not (unshaked ?s))
(not (contains ?s ?d1))
(not (contains ?s ?d2))
(shaked ?s)
(contains ?s ?b)))

(:action pour-shaker-to-shot
:parameters (?b - beverage ?d - shot ?h -
hand ?s - shaker ?l ?l1 - level)
:precondition (and (holding ?h ?s)
(shaked ?s)
(empty ?d)
(clean ?d)
(contains ?s ?b)
(shaker-level ?s ?l)
(next ?l1 ?l))
:effect (and (not (clean ?d))
(not (empty ?d))
(contains ?d ?b)
(shaker-level ?s ?l1)
(not (shaker-level ?s ?l))))
)
```

The $\mathbb{DF}$ contains all twelve actions and their pre-conditions and post-conditions, as well as predicates needed for the domain.

$\mathbb{PF}$:

```
(define (problem prob)
(:domain barman)
(:objects
shaker1 - shaker
left right - hand
shot1 - shot
ingredient1 ingredient2 - ingredient
cocktail1 - cocktail
dispenser1 dispenser2 - dispenser
l0 l1 l2 - level
)
(:init
(ontable shaker1)
(ontable shot1)
(dispenses dispenser1 ingredient1)
(dispenses dispenser2 ingredient2)
(clean shaker1)
(clean shot1)
(empty shaker1)
(empty shot1)
(handempty left)
(handempty right)
(shaker-empty-level shaker1 l0)
(shaker-level shaker1 l0)
(next l0 l1)
(next l1 l2)
(cocktail-part1 cocktail1 ingredient1)
(cocktail-part2 cocktail1 ingredient2)
)
(:goal
(and
(contains shot1 cocktail1)
)))
```

The $\mathbb{PF}$ contains the objects, initial state and goal state for the problem.

A.8 `Barman-100` $\mathbb{DD}$ and $\mathbb{PD}$

The following display example $\mathbb{DD}$ and $\mathbb{PD}$ for the heavily templated data for `Barman-100`. We can see that the descriptions have all the same components as the $\mathbb{DF}$ and $\mathbb{PF}$ in PDDL.

For the Heavily Templated $\mathbb{DD}$, all preconditions and post-conditions are written out explicitly and sound similar to PDDL. The $\mathbb{PD}$ is similar, in that it lists all the predicates needed for to solve the task.

Heavily Templated $\mathbb{DD}$:

```
I am creating a cocktail from a set of
ingredients. Here are the actions I can do

Grasp a container (grasp hand container)
Leave a container (leave hand container)
Fill a shot glass with with an ingredient
(fill-shot shot ingredient hand1 hand2
dispenser)
Re-fill a shot glass with an ingredient
(refill-shot shot ingredient hand1 hand2
dispenser)
Empty a shot glass (empty-shot hand shot
beverage)
Clean a shot glass (clean-shot shot
beverage hand hand2)
Pour an ingredient from a shot glass to
a clean shaker (pour-shot-to-clean-shaker
shot ingredient shaker hand level level1)
Pour an ingredient from a shot glass to a
used shaker (pour-shot-to-used-shaker shot
ingredient shaker hand level level1)
Empty a shaker (empty-shaker hand shaker
cocktail level level1)
Clean a shaker (clean-shaker hand1 hand2
shaker)
Shake a shaker (shaker cocktil ingredient1
ingredient2 shaker hand1 hand2)
Pour a cocktail from a shaker to a shot
glass (pour-shaker-to-shot beverage shot
hand shaker level level1)
I have the following restrictions on my
actions:
To perform Grasp action, the following
facts need to be true: container on table,
hand empty.
Once Grasp action is performed the
following facts will be true: hand holding
container.
Once Grasp action is performed the
following facts will be false: container
on table, hand empty.
To perform Leave action, the following
facts need to be true:  hand holding
container.
Once Leave action is performed the
following facts will be true: hand empty,
container on table.
Once Leave action is performed the
following facts will be false:  hand
holding container.
To perform Fill-shot action, the following
needs to be true:  hand1 holding shot
glass, hand2 empty, dispenser dispenses
ingredient, empty shot glass, clean shot
glass.
Once Fill-shot action is performed the
following will be true:  shot glass
contains ingredient, shot glass used with
ingredient.
Once Fill-shot action is performed the
following will be false: empty shot glass,
clean shot glass.
```

To perform Refill-shot action, the following needs to be true: hand1 holding shot glass, hand2 empty, dispenser dispenses ingredient, empty shot glass, shot glass used with ingredient.
Once Refill-shot action is performed the following will be true: shot glass contains ingredient.
Once Refill action is performed the following will be false: empty shot glass.
To perform Empty-shot action, the following needs to be true: hand holding shot glass, shot glass contains beverage.
Once Empty-shot action is performed the following will be true: empty shot glass.
Once Empty-shot action is performed the following will be false: shot glass contains beverage.
To perform Clean-shot action, the following needs to be true: hand1 holding shot glass, hand2 empty, empty shot glass, shot glass used with beverage.
Once Clean-shot action is performed the following will be true: clean shot glass.
Once Clean-shot action is performed the following will be false: shot glass used with beverage
To perform Pour-shot-to-clean-shaker action, the following needs to be true: hand1 holding shot glass, shot glass contains ingredient, empty shaker, clean shaker, shaker-level of shaker is l, next level from l is l1.
Once Pour-shot-to-clean-shaker action is performed the following will be true: empty shot
glass, shaker contains ingredient, unshaked shaker, shaker-level of shaker is l1.
Once Pour-shot-to-clean-shaker action is performed the following will be false: shot glass contains ingredient, empty shaker, clean shaker, shaker-level of shaker is l.
To perform Pour-shot-to-used-shaker action, the following needs to be true: hand1 holding shot glass, shot glass contains ingredient, unshaked shaker, shaker-level of shaker is l, next level from l is l1.
Once Pour-shot-to-used-shaker action is performed the following will be true: shaker contains ingredient, empty shot glass, shaker-level of shaker is l1.
Once Pour-shot-to-used-shaker action is performed the following will be false: shot glass contains ingredient, shaker-level of shaker is l.

To perform Empty-shaker action, the following needs to be true: hand holding shaker, shaker contains cocktail, shaked shaker, shaker-level of shaker is l, shaker-empty-level of shaker is l1.
Once Empty-shaker action is performed the following will be true: shaker-level of shaker is l1, empty shaker.
Once Empty-shaker action is performed the following will be false: shaked shaker, shaker-level of shaker is l, shaker contains cocktail.
To perform Clean-shaker action, the following needs to be true: hand1 holding shaker, hand2 empty, empty shaker.
Once Clean-shaker action is performed the following will be true: clean shaker.
To perform Shake action, the following needs to be true: hand1 holding shaker, empty hand2, shaker contains ingredient1, shaker contains ingredient2, part 1 of cocktail is ingredient1, part 2 of cocktail is ingredient2, unshaked shaker.
Once Shake action is performed the following will be true: shaked shaker, shaker contains cocktail.
Once Shake action is performed the following will be false: unshaked shaker, shaker contains ingredient1, shaker contains ingredient2.
To perform Pour-shaker-to-shot action, the following needs to be true: hand holding shaker, shaked shaker, empty shot glass, clean shot glass, shaker contains cocktail, shaker level of shaker is l, next level from l1 is l.
Once Pour-shaker-to-shot action is performed the following will be true: shot glass contains cocktail, shot glass used with cocktail, shaker-level of shaker is l1.
Once Pour-shaker-to-shot action is performed the following will be false: clean shot glass, empty shot glass, shaker-level of shaker is l.

$\mathbb{DD}$ (cont'd)

Heavily Templated $\mathbb{PD}$:

```
For this cocktail, I have the following: shaker shaker1, my left hand, my right hand, shot glass shot1, ingredient ingredient1, ingredient ingredient2, dispenser dispenser1, and dispenser dispenser2. The shaker has the following levels: l0, l1, and l2. I want to make the following cocktails: cocktail1.
As initial conditions, I have that shaker1 is on the table, shot1 is on the table, dispenser1 dispenses ingredient1, dispenser2 dispenses ingredient2, shaker1 is clean, shot1 is clean, shaker1 is empty, shot1 is empty, handempty left, handempty right, shaker-empty-level shaker1 l0, shaker-level shaker1 l0, next l0 l1, next l1 l2, cocktail-part1 cocktail1 ingredient1, and cocktail-part2 cocktail1 ingredient2.
My goal is to have that shot1 contains cocktail1.
```

B Prompts

For the LLM-as-planner, we give all the models the following prompt for `BlocksWorld-100`:

```
Here is a game involving a table with blocks on it.

{domain_description}

{problem_description}

Write the plan that would solve this problem.

These are the available actions:
(PICK-UP block): pick up a block from the table
(PUT-DOWN block): put down a block on the table
(STACK block1 block2): stack block1 onto block2
(UNSTACK block1 block2): unstack block1 from block2

Here is what the output should look like:
(PICK-UP A)
(STACK A B)
(UNSTACK A B)
(PUT-DOWN A)
```

For `MysteryBlocksWorld-100`, we use the following prompt:

```
Here is a game involving a table with blocks on it.

{domain_description}

{problem_description}

Write the plan that would solve this problem.

These are the available actions:
(ATTACK object): attack object
(SUCCUMB object): succumb
(OVERCOME object1 object2): overcome object1 from object2
(FEAST object1 object2): feast object1 from object2

Here is what the output should look like:
(ATTACK A)
(OVERCOME A B)
(FEAST A B)
(SUCCUMB A)
```

Whenever possible, we asked the model to return the output in a JSON object for easier parsing.

C Detailed Results

Beyond the visualizations above, we show the detailed results of all models on all simulations of all naturalness levels.

C.1 Results for `BlocksWorld-100`

Table 3 displays results for all results for `BlocksWorld-100`.

C.2 Results for `MysteryBlocksWorld-100`

Table 4 displays the results for Heavily Templated `MysteryBlocksWorld-100`.

C.3 Results for `Logistics-100`

Table 5 displays results for all naturalness settings for `Logistics-100`

C.4 Results for `Barman-100`

Table 6 displays results for `Barman-100`.

Metrics		
	Natural	
Models	Solvability	Correctness
gemma-2-9b-it	3/100	3/100
gemma-2-9b-it^p	-	9/100
gemma-2-27b-it	0/100	0/100
gemma-2-27b-it^p	-	11/100
Llama-3.1-8B	0/100	0/100
Llama-3.1-8B^p	-	1/100
Llama-3.1-70B	0/100	0/100
Llama-3.1-70B^p	-	13/100
gpt-3.5-turbo	2/100	1/100
gpt-4o-mini	19/100	3/100
gpt-4o-mini^p	-	7/100
gpt-4o	64/100	60/100
gpt-4o^p	-	33/100
o1-preview	91/100	91/100
o1-preview^p	-	82/100
o3-mini	79/100	68/100
o3-mini^p	-	87/100
	Moderately Templated	
Models	Solvability	Correctness
gemma-2-9b-it	0/100	0/100
gemma-2-9b-it^p	-	5/100
gemma-2-27b-it	17/100	10/100
gemma-2-27b-it^p	-	3/100
Llama-3.1-8B	0/100	0/100
Llama-3.1-8B^p	-	1/100
Llama-3.1-70B	0/100	0/100
Llama-3.1-70B^p	-	10/100
gpt-3.5-turbo	14/100	4/100
gpt-4o-mini	9/100	5/100
gpt-4o-mini^p	-	7/100
gpt-4o	77/100	67/100
gpt-4o^p	-	35/100
o3-mini	82/100	70/100
o3-mini^p	-	87/100
	Heavily Templated	
Models	Solvability	Correctness
gemma-2-9b-it	61/100	10/100
gemma-2-9b-it^p	-	5/100
gemma-2-27b-it	81/100	80/100
gemma-2-27b-it^p	-	7/100
Llama-3.1-8B	0/100	0/100
Llama-3.1-8B^p	-	0/100
Llama-3.1-70B	0/100	0/100
Llama-3.1-70B^p	-	10/100
gpt-3.5-turbo	39/100	29/100
gpt-4o-mini	66/100	59/100
gpt-4o-mini^p	-	1/100
gpt-4o	89/100	89/100
gpt-4o^p	-	29/100
o3-mini	94/100	94/100
o3-mini^p	-	96/100

Table 3: Performance of LLM-as-formalizer and LLM-as-planner (^{p}) all BlocksWorld-100 data.

Models	Solvability	Correctness
Llama-3.1-8B	0/100	0/100
Llama-3.1-8B^p	-	0/100
Llama-3.1-70B	0/100	0/100
Llama-3.1-70B^p	-	0/100
gemma-2-9b-it	100/100	99/100
gemma-2-9b-it^p	-	9/100
gemma-2-27b-it	99/100	98/100
gemma-2-27b-it^p	-	0/100
gpt-3.5-turbo	4/100	0/100
gpt-4o-mini	36/100	5/100
gpt-4o-mini^p	-	0/100
gpt-4o	74/100	70/100
gpt-4o^p	-	0/100
o3-mini	95/100	95/100
o3-mini^p	-	74/100

Table 4: Performance of LLM-as-formalizer and LLM-as-planner (^{p}) on the Heavily Templated MysteryBlocksWorld-100.

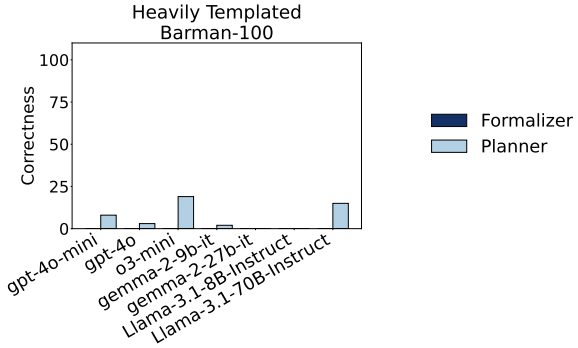

Figure 5: Performance for Barman-100.

D Sample Model Output

The following is an example $\mathbb{DF}$ and $\mathbb{PF}$ that Llama-3.1-8B-Instruct gave. We can see that there are syntax errors, as well as semantic errors in the $\mathbb{DF}$ and $\mathbb{PF}$.

Metrics		
	Natural	
Models	Solvability	Correctness
gemma-2-9b-it	1/100	0/100
gemma-2-9b-it^p	-	0/100
gemma-2-27b-it	1/100	0/100
gemma-2-27b-it^p	-	0/100
Llama-3.1-8B	0/100	0/100
Llama-3.1-8B^p	-	1/100
Llama-3.1-70B	0/100	0/100
Llama-3.1-70B^p	-	0/100
gpt-3.5-turbo	1/100	0/100
gpt-3.5-turbo^p	-	0/100
gpt-4o-mini	13/100	0/100
gpt-4o-mini^p	-	0/100
gpt-4o	20/100	2/100
gpt-4o^p	-	1/100
o3-mini	43/100	7/100
o3-mini^p	-	14/100
	Moderately Templated	
Models	Solvability	Correctness
gemma-2-9b-it	0/100	0/100
gemma-2-9b-it^p	-	0/100
gemma-2-27b-it	3/100	0/100
gemma-2-27b-it^p	-	0/100
Llama-3.1-8B	0/100	0/100
Llama-3.1-8B^p	-	0/100
Llama-3.1-70B	0/100	0/100
Llama-3.1-70B^p	-	0/100
gpt-3.5-turbo	0/100	0/100
gpt-3.5-turbo^p	-	0/100
gpt-4o-mini	29/100	0/100
gpt-4o-mini^p	-	0/100
gpt-4o	34/100	13/100
gpt-4o^p	-	0/100
o3-mini	51/100	39/100
o3-mini^p	-	21/100
	Heavily Templated	
Models	Solvability	Correctness
gemma-2-9b-it	1/100	0/100
gemma-2-9b-it^p	-	0/100
gemma-2-27b-it	0/100	0/100
gemma-2-27b-it^p	-	1/100
Llama-3.1-8B	0/100	0/100
Llama-3.1-8B^p	-	0/100
Llama-3.1-70B	0/100	0/100
Llama-3.1-70B^p	-	0/100
gpt-3.5-turbo	1/100	1/100
gpt-3.5-turbo^p	-	0/100
gpt-4o-mini	26/100	0/100
gpt-4o-mini^p	-	0/100
gpt-4o	33/100	13/100
gpt-4o^p	-	0/100
o3-mini	55/100	47/100
o3-mini^p	-	21/100

Table 5: Performance of LLM-as-formalizer and LLM-as-planner (^{p}) all Logistics-100 data.

We give the model the following input $\mathbb{DD}$ and $\mathbb{PD}$:

Models	Solvability	Correctness
Llama-3.1-8B	0/100	0/100
Llama-3.1-8B^p	-	0/100
Llama-3.1-70B	0/100	0/100
Llama-3.1-70B^p	-	15/100
gemma-2-9b-it	0/100	0/100
gemma-2-9b-it^p	-	2/100
gemma-2-27b-it	0/100	0/100
gemma-2-27b-it^p	-	0/100
gpt-3.5-turbo	0/100	0/100
gpt-3.5-turbo^p	-	4/100
gpt-4o-mini	0/100	0/100
gpt-4o-mini^p	-	8/100
gpt-4o	1/100	0/100
gpt-4o^p	-	3/100
o3-mini	1/100	0/100
o3-mini^p	-	19/100

Table 6: Performance of LLM-as-formalizer and LLM-as-planner (^{p}) on the Heavily Templated Barman-100.

```
Blocksworld is a puzzle domain where the
aim is to move a set of blocks to achieve
a specific goal configuration. The domain
consists of a flat surface, often called
the 'table', and an arbitrary number of
blocks. Every block can be stacked on top
of another block or placed on the table.
Blocks can be moved one at a time, and
only the block at the top of a stack can
be moved. The main actions available in
Blocksworld are:

1. **Pick Up (Pick-Up B)**: Picks up
block B from the table.
2. **Put Down (Put-Down B)**: Places block
B onto the table.
3. **Stack (Stack B1 B2)**: Stacks block
B1 on top of block B2.
4. **Unstack (Unstack B1 B2)**: Removes
block B1 from the top of block B2.

The goal is to transform an initial
configuration of blocks into a desired
target configuration using these actions.

In this Blocksworld problem, we will
consider a simple scenario with 2 blocks:
Block1 and Block2. The initial state of
the blocks is as follows:
**Initial State**:
- Block1 is on the table.
- Block2 is on Block1.

The goal is to achieve the following
configuration:

*Goal State**:
- Block2 is on the table.
- Block1 is on the table.

To solve this problem, you will need
to use the available actions to rearrange
the blocks from the initial state to the
goal state.
```

Which returns the following Output $\mathbb{DF}$:

```
(define (domain blocksworld)
(:requirements :strips :typing)
(:types block)
(:predicates
(on-table?b - block)
(on?b1 - block?b2 - block)
(clear?b - block)
)
 (:action pick-up
:parameters (?b - block)
:preconditions      (and      (on-table?b)
(clear?b))
:effects  (and  (not  (on-table?b))  (not
(clear?b)) (on-table?b))
)
(:action put-down
:parameters (?b - block)
:preconditions  (and  (not  (on-table?b))
(clear?b))
:effects (and (on-table?b) (clear?b))
)
(:action stack
:parameters (?b1 - block?b2 - block)
:preconditions      (and      (on-table?b1)
(clear?b2))
:effects  (and  (not  (on-table?b1))  (not
(clear?b2)) (on?b1?b2))
)
(:action unstack
:parameters (?b1 - block?b2 - block)
:preconditions (and (on?b1?b2) (clear?b1))
:effects    (and    (on-table?b1)    (not
(on?b1?b2)) (clear?b2))
)
)
```

Output $\mathbb{PF}$:

```
(define (problem blocksworld-problem)
(:domain blocksworld)
(:objects block1 block2 - block)
(:init
(on-table block1)
(on block2 block1)
(clear block1)
(clear block2)
)
(:goal
(and (on-table block2)
(on-table block1)
(clear block1)
(clear block2)
)
)
)
```

Text in red displays errors outputted from the model. We can see that in the $\mathbb{DF}$ there are syntax errors (incorrect keyword "preconditions") as well as semantic errors (incorrect predicates in preconditions and effects). For the output $\mathbb{PF}$ there are semantic errors, such as incorrect preconditions (a block cannot be clear and have another block on top of it) in the init section.

E Related Works Comparison

Table 7 compares works related to this paper. We can see that other works as the LLM to predict either the plan, parts of PDDL files and other languages. We can also see that other works have mostly templated natural language descriptions, while this work uses both templated and natural descriptions.

F Methodology Comparison

Both methodologies used in this paper incorporate LLMs into planning. There are pros and cons to each methodology. When using LLM-as-planner, we have a lightweight solution that returns results very quickly. However, due to the lack of reasoning skills in LLMs, they often struggle to create formal plans. Meanwhile using LLM-as-formalizer provides better executability and interpretability, though it uses a solver, which may results in getting results slower. We believe that for performance reasons that LLM-as-formalizer is the superior methodology.

	Environment	LLM predicts?	Natural Descriptions?
Zuo et al. (2024)	fully-observed	$\mathbb{PF}$	N
Zhang et al. (2024a)	partially-observed	$\mathbb{PF}$	N
Liu et al. (2023a)	fully-observed	$\mathbb{PF}$	N
Xie et al. (2023)	fully-observed & partially-observed	$\mathbb{PF}$ goal	N
Lyu et al. (2023)	fully-observed	$\mathbb{PF}$ goal	N
Zhang et al. (2024c)	procedural texts	$\mathbb{DF}$ action semantics	N
Wong et al. (2023)	partially-observed	$\mathbb{DF}$	N
Guan et al. (2023)	fully-observed	$\mathbb{DF}$ & $\mathbb{PF}$ goal	N
Zhu et al. (2024)	fully-observed	$\mathbb{DF}$ action semantics	N
Tang et al. (2024)	partially-observed	Python	N/A
Silver et al. (2024)	fully-observed	plan	N
Valmeekam et al. (2024)	fully-observed	plan	N
Stechly et al. (2024)	fully-observed	plan	N
This work	fully-observed	$\mathbb{DF}$ & $\mathbb{PF}$	Y

Table 7: Comparison with related work.