Language Model as Planner and Formalizer under Constraints

Anonymous ACL submission

Abstract

LLMs have been widely used in planning, either as planners to generate action sequences end-to-end, or as formalizers to represent the planning domain and problem in a formal language that can derive plans deterministically. However, both lines of work rely on standard benchmarks that only include generic and simplistic environmental specifications, leading to potential overestimation of the planning ability of LLMs and safety concerns in downstream tasks. We bridge this gap by augmenting widely used planning benchmarks with manually annotated, fine-grained, and rich natural language constraints spanning four formally defined categories. Over 4 state-of-the-art reasoning LLMs, 4 formal languages, and 4 datasets, we show that the introduction of one-sentence constraints consistently halves performance, indicating current LLMs' lack of robustness and avenue for future research.[1]

1 Introduction

Large language models (LLMs) have garnered attention in recent years for their capabilities in planning domains. An intuitive methodology, *LLM-as-planner*, directly generates a plan given a description of the domain and the problem in an end-to-end manner. While the algorithmic nature of formal planning tasks significantly challenge LLM planners (Valmeekam et al., 2024a; Kambhampati et al., 2024), the emergence of reasoning LLMs has shown improved performance (Valmeekam et al., 2024b; Huang and Zhang, 2025) but lacks interpretability and control. Another methodology, *LLM-as-formalizer*, instead translates the input description into some formal languages (Xie et al., 2023; Liu et al., 2023a; Zhang et al., 2024a,b; Zhu et al., 2024). This formal representation can then be passed into a solver which then outputs a plan deterministically, providing more interpretability, control, and promising performance.

As a recent, nascent research direction, most previous work relies on standard planning benchmarks, some having existed for decades, where environments are described in a generic, homogeneous, simplistic manner. This is a critical issue because the simplicity of the data may lead to overestimation of the planning or formalizing ability of an LLM. While existing benchmarks enable proof of concept, they are distant from real-life planning instructions that commonly include idiosyncratic requirements inflicted by the user or the resources.

To bridge this gap, we introduce `CoPE` (**Co**nstrained **P**lanning **E**nvironments), a benchmark that augments widely used planning environments such as BlocksWorld (IPC, 1998) and CoinCollector (Yuan et al., 2019) with **constraints**. Each example originally includes a domain description, a problem description, and ground-truth PDDL as a simulator to validate a predicted plan. In `CoPE`, we additionally annotate a natural language constraint that is not only linguistically rich but also formally categorized (Figure 1). We systematically evaluate both LLM-as-planner and LLM-as-formalizer methodologies, the latter including 4 formal languages including Planning Definition Language (PDDL) 1.2 and 3, Linear Temporal Logic (LTL), and Satisfiability Modulo Theories (SMT). Among 4 state-of-the-art LLMs on 4 datasets, we show that one-sentence constraints consistently halves performance, indicating lack of robustness of both LLM planning methodologies. To inspire future work, we show that different formalisms respond differently to categories of constraints and also domains. As we corroborate past work by showing the robustness of LLM-as-formalizer over LLM-as-planner to problem complexity and lexical shift, we clearly demonstrate that the introduction of constraints substantially negates such robustness. We pose our findings as a timely reflection among a

[1] Our code and data are attached with the submission.

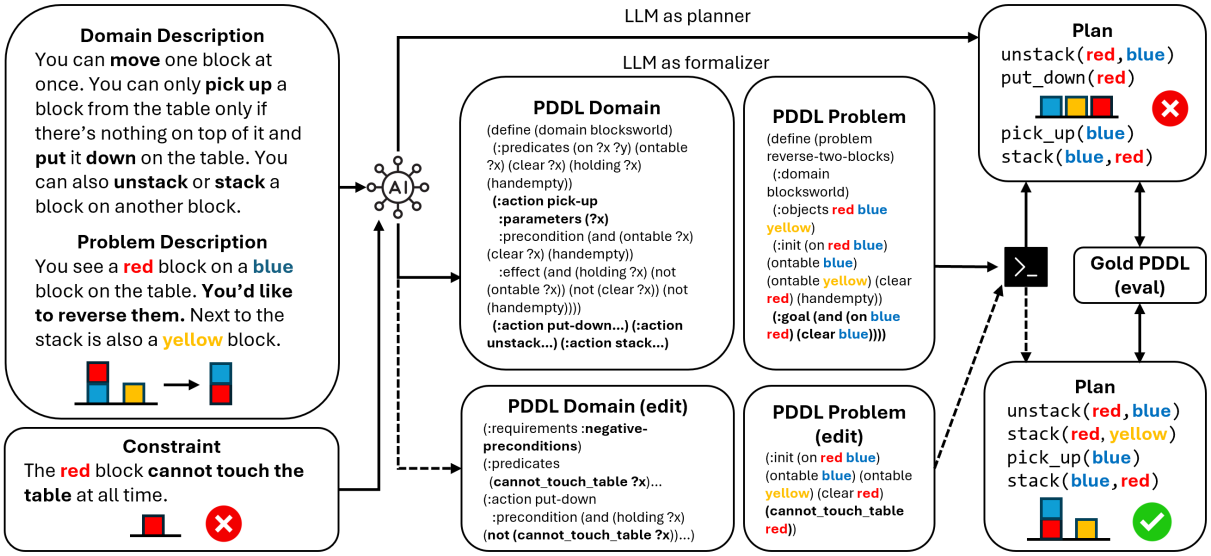

Figure 1: The flow of evaluation on CoPE. An LLM is given a description of the planning domain and problem, as well as a natural language constraint. As a planner, the LLM directly outputs a plan consisting of a sequence of actions. As a formalizer, the LLM attempts to construct a formal representation that can invoke an external solver to derive a plan. Alternatively, to address the constraint, the LLM as an editing formalizer (dashed arrows) would first predict PDDL without the constraint before generating edits to it.

recent slew of methodological work in LLM-based planning, highlighting limitations and opening up future directions.

2 Related Work

As comprehensive literature reviews of the LLM-as-planner and LLM-as-formalizer methodology have been accomplished by recent surveys (Tantakoun et al., 2025; Wei et al., 2025), we will instead focus on the benchmarks used in existing work.

Planning Benchmarks Most work on planning with LLMs experimented with International Planning Competition (IPC) domains including BlocksWorld, Gripper, Logistics, Barman, etc. (Kagitha et al., 2025; Hu et al., 2025; Zhu et al., 2024; Liu et al., 2023a; Shirai et al., 2024; Hao et al., 2025; Guo et al., 2024; Shojaee et al., 2025; Stein et al., 2025; Stechly et al., 2024; Valmeekam et al., 2024a; Yao et al., 2025; Guan et al., 2023). We provide a table of works and a non-exhaustive list the domains used, as well as examples from common domains in Appendix A. Beyond those domains and datasets that have existed for decades, few planning benchmarks have been constructed in recent years. Moreover, Zuo et al. (2024); Huang and Zhang (2025) recently pointed out the templated nature and linguistic homogeneity of these datasets and attempted to describe the domains and problems in a natural fashion. While this modification added difficulty to the task, the semantics of the domains and problems remain standard, subject to LLMs' memorization, and does not reflect the complexity of real-world planning.

Data Augmentation Much recent work augmented datasets for different tasks in order to make the problem more realistic, challenging, and less prone to data contamination. This has been done for question answering (Anantheswaran et al., 2024; Hong et al., 2024), common sense reasoning (Sun and Emami, 2024), instruction following (Hong et al., 2024), and so on. For planning, datasets have been augmented to increase robustness, for example by adding noise (Zheng et al., 2025) and lexical perturbation (Valmeekam et al., 2024a; Zheng et al., 2024), with no change to the actual semantics. In contrast, our proposed CoPE semantically modifies both the domain and the problems, leading to increased realism and challenge.

Planning With Constraints In the formal planning community, syntax that expresses some constraints were introduced to the PDDL3 language (Gerevini and Long, 2005; Edelkamp, 2006). However, few follow-up work in the NLP and AI community recognized and addressed the real-life challenge of planning with constraints. Yang et al. (2023) introduced a hybrid system that translates natural language constraints into LTL to assist LLM-as-planner methods, though their focus was

primarily on robotic applications and lacked a thorough discussion of diverse constraint patterns and formal representations. Guo et al. (2024) encoded constraints with code to interface a SMT solver and combined it with PDDL for task and motion planning though without any public data or code. Moreover, they only considered coarse-grained constraints and oversimplified the task by assuming parts of the PDDL which is often not realistic. We closely consider these formalisms when defining the constraints.

3 Task Formulation

Definition of Planning We will first formally define the task of text-based formal planning in line with classical STRIPS formalism (Fikes and Nilsson, 1971) and recent conventions in the natural language processing community (Huang and Zhang, 2025). The input I consists of a triplet $(D_d, D_p, \mathcal{DF}')$:

1. D_d is a textual description of the domain, specifying the types (the type system for entities e_i), predicates (in relation to the types), and actions a_i including pre-conditions and effects (parameterized by conjunctive logical formulas over the the predicates).
2. D_p is a textual description of the problem, specifying the objects, initial states and goal states in terms of predicates;
3. $\mathcal{DF}'_{\mathcal{G}}$ is the header of the ground-truth PDDL domain file $\mathcal{DF}_{\mathcal{G}}$ that only provides the names and parameters of the actions. This is required so that the eventual plan is grounded to an existing ontology and can be evaluated fairly.

The final objective is a plan $\pi = [a_0(\bar{e}_0), \ldots, a_m(\bar{e}_m)]$, where each $a_i \in A$ is an action provided by $\mathcal{DF}'_{\mathcal{G}}$, and each $\bar{e}_i$ is a tuple of grounded entities corresponding to the parameters of a_i. The plan π, when executed in the environment expressed by a ground-truth $\mathcal{DF}_{\mathcal{G}}$ and $\mathcal{PF}_{\mathcal{G}}$, must (i) fulfill the pre-conditions of each action, and (ii) constitutes a successful transition from the initial state to the goal state. We will revisit the formalism of plan execution after defining the domain and problem files.

The domain file $\mathcal{DF}$ is defined by three main components: types (T), predicates (R), and action semantics ($a^{param}, \psi_a^{pre}, \psi_a^{eff}$):

1. Types: $T = t_1, \ldots, t_n$ is the set of entity types present in the environment.
2. Predicates: $R = r_1, \ldots, r_n$ is the set of relational predicates, each in the form of $r(\bar{t})$, where $\bar{t}$ is a tuple of types that participate in relation r.
3. Action semantics: given an action, the parameters a^{param} are a set of types $\bar{t}$ that participate in this action. The pre-conditions $\psi_a^{pre} : \mathcal{S} \to \mathbb{B}$ is a boolean formula over the current state which presides over a set of predicates $\bar{r}$, where S is the state space. The action is executable iff $\psi_a^{pre} = \text{True}$. The effects $\psi_a^{eff} : \mathcal{S} \to \mathcal{S}$ is a formula that transition the current state to the next state after the execution of this action.

The problem file $\mathcal{PF}$ is defined by three main components: objects (E), the initial state (s_0), and the goal states ($\mathcal{S}_g$):

1. Objects: $E = e_1, \ldots, e_n$ is the set of named entities present in the environment. Each object is an instance of an entity type t defined in $\mathcal{DF}'_{\mathcal{G}}$.
2. Initial State: $s_0 \in \mathcal{S}$ is a state defined by set of relational facts of the form $r(\bar{e})$, where $r \in R$ is a relational predicate; $\bar{e}$ is a tuple of entities from E that participate in relation r.
3. Goal States: $\mathcal{S}_g \subseteq \mathcal{S}$ specifies the set of goal states to be reached. A state s is a goal state if and only if $s \in \mathcal{S}_g$.

Here, the state space $\mathcal{S}$ comprises all possible configurations of relational facts over the object set E with the predicates R. Each state $s \in \mathcal{S}$ is a set of instantiated facts $\bar{r}(\bar{e})$, describing which relations hold among the objects. Successfully executing a plan $\pi = [a_0(\bar{e}_0), \ldots, a_m(\bar{e}_m)]$ from the initial state yields a trace of states $L = [s_0, s_1, \ldots, s_{m+1}]$, where $\psi_{a_i}^{pre}(s_i) = \text{True}$, $s_{i+1} = \psi_{a_i}^{eff}(s_i)$, and $s_{m+1} \in \mathcal{S}_g$.

While LLM-as-planner maps the input triplet I directly to a plan L, LLM-as-formalizer generates the domain file $\mathcal{DF}_{\mathcal{P}}$ and $\mathcal{PF}_{\mathcal{P}}$ before inputting them into a PDDL solver to search for a plan.

Definition of Constraint While the definition of *constraint* varies by community, we take a linguistic and pragmatic approach and define it as supplementary, non-destructive information to the task description that restricts possible behaviors in a plan. We aim to power real-world systems that can adhere to any constraint expressed by human users, encompassing formal definitions from established fields (e.g., classical planning) without being limited by them. Specifically, `CoPE` includes formally categorized constraints as follows.

Category	Example constraint	Formal explanation	Plan
None			unstack(red,blue) put_down(red) pick_up(blue) stack(blue,red)
Initial	The red, blue and yellow blocks are on the table.	$s'_0 = \{$on-table(red) $\wedge$ on-table(blue) $\wedge$ on-table(yellow)$\}$	pick_up(blue) stack(blue,red)
Goal	You'd like to have all three blocks on the table.	$\mathcal{S}'_g \models \{$on-table(red) $\wedge$ on-table(blue) $\wedge$ on-table(yellow)$\}$	unstack(red,blue) put_down(red)
Action	If you move the red block you must move the yellow block immediately after.	$\Pi - \Pi' = \{L \mid \exists\, a_i \in L,$ $a_i =$ move(red)$\wedge$ $a_{i+1} \neq$ move(yellow), $L \in \mathcal{L}\}$	unstack(red,blue) put_down(red) pick_up(yellow) put_down(yellow) pick_up(blue) stack(blue,red)
State	Once you start, the number of blocks in each stack should not exceed 1 at any time.	$\mathcal{L} - \mathcal{L}' = \{s \mid \forall\, b \in \bar{b},$ height(b) $> 1, s \in \mathcal{S}\}$	Plan does not exist

Table 1: Examples of constraints in each category categories, explanation of the assignment, and how they would affect the plan of an example problem in BlocksWorld (Figure 1): "You see a red block on a blue block on the table. You'd like to reverse them. Next to the stack is also a yellow block."

We consider a short text $\mathcal{C}$ as a constraint when a plan is valid if and only if it satisfies each of the conditions specified by $\mathcal{C}$. We introduce two concepts *primitive* and *modified*, where applying constraints modifies the primitive action or state space.

Definition 1. *An action is a **primitive action** a^* of an action a, if $\psi_a^{pre} \models \psi_{a^*}^{pre}, \psi_a^{eff} \models \psi_{a^*}^{eff}$.*

Definition 2. *A state is a **primitive state** s^* of a state s, if $s \models s^*$.*

Assuming constraints only introduce new predicates but do not remove any existing ones, every modified action will find exactly one primitive action, and every modified state will find exactly one primitive state from the original action and state space, respectively. We then define the constraint types as follows:

Definition 3. *Let $L'^* = [s'^*_0, \ldots, s'^*_T]$ denote a trace of primitive states after introducing $\mathcal{C}$, and s^*_0 be initial states without $\mathcal{C}$, $\mathcal{C}$ is an **initial constraint** iff $s'^*_0 = s'_0$ (no new predicate is introduced to the initial state) and $s'^*_0 \neq s^*_0$ (the initial state changes after introducing $\mathcal{C}$).*

Definition 4. *Let $\mathcal{S}_g$ and $\mathcal{S}'_g$ be goal states before and after introducing $\mathcal{C}$, $\mathcal{C}$ is a **goal constraint** iff $\{s \mid s \in \mathcal{S}_g, s \in \mathcal{S}\} \neq \{s'^* \mid s' \in \mathcal{S}'_g, s' \in \mathcal{S}'\}$.*

Definition 5. *Let $\pi'^* = [a'^*_0, \ldots, a'^*_T]$ denote a sequence of primitive actions after introducing $\mathcal{C}$, and Π and Π' be all valid plans before and after introducing $\mathcal{C}$, $\mathcal{C}$ is an **action constraint** iff $\Pi'^* \subsetneq \Pi^*$, where Π'^* is a collection of all π'^*, $\pi' \in \Pi'$.*

Definition 6. *Let $L' = [s'^*_0, \ldots, s'^*_T]$ denote a trace of primitive states after introducing $\mathcal{C}$, and $\mathcal{L}$ and $\mathcal{L}'$ be all valid state traces before and after introducing $\mathcal{C}$, $\mathcal{C}$ is an **state constraint** iff $\mathcal{L}'^* \subsetneq \mathcal{L}$, where $\mathcal{L}'^*$ is a collection of all L'^*, $L' \in \mathcal{L}'$, and $\mathcal{C}$ is not any other constraint defined above.*

Our categorization is complete as the definition of state constraints subsumes all possible constraints. By design, each category of constraints is expressed differently for each formalism (PDDL, PDDL3, LTL, SMT), requiring different level of modification to the original code. Notably, the union of the four categories is not trivialized by any formalism. For example, simply using the the *constraint* syntax in PDDL3 cannot express action constraints or state constraints requiring invention of new predicates. While conjunction of multiple predicates is possible, we do not discuss it in this work. Table 1 shows examples of constraints and justification of their categories.

4 Data

To evaluate LLM planning in environments with constraints, we propose the CoPE benchmark that includes two widely used planning domains:

BlocksWorld (IPC, 1998) is a domain to rearrange stacks of blocks on a table using a robotic arm. It

involves four actions: *pick up*, *put down*, *stack*, and *unstack*. We build on the `BlocksWorld-100` benchmark from Huang and Zhang (2025) with 100 problems, each including a domain and problem description as well as the headers of actions $(D_d, D_p, \mathcal{DF}')$ as input to models. It also comes with ground-truth PDDL $\mathcal{DF}$ and $\mathcal{PF}$, which are used to validate a predicted plan.

CoinCollector (Yuan et al., 2019) is a domain to navigate a house full of rooms in order to find a coin. It involves two actions: *move* and *pick up*. Originally designed to evaluate interactive agents, this is a partially observable environment where no complete information is given and so making a complete plan is impossible. To place it under the framework of `CoPE`, we convert it into a fully observable environment by Using a depth-first search, we find all room arrangements and create problem descriptions and ground-truth problem files. We sample 20 problems with 3, 5, 7, 9, and 11 rooms using the benchmark from Jansen and Côté (2022). In total, we also have 100 problem instances consisting of input and ground-truth PDDL.

While these two domains are likely included in modern LLMs' training data, they remain standard in literature and meaningfully challenge state-of-the-art LLMs. `CoPE` itself is a step forward to address memorization by changing the semantics of the domains and problems. For each domain, we manually annotate 100 constraints spanning the four categories with ground-truth code of each formalism. Unlike the original datasets, we allow problems to only occasionally have no solutions. In the full set of `CoPE`, each constraint is paired with each problem, resulting in 10,000 constrained planning problems. To evaluate the most representative cases while keeping analysis tractable, we our following evaluation on a subset where we manually pair each constraint and one representative problem, leading to 100 problems per domain. Examples are shown in Appendix B.

5 Experimental Setup

Given the input with the natural language constraint $(D_d, D_p, \mathcal{DF}', \mathcal{C})$, we prompt the LLM to generate 4 types of outputs: a plan directly (**LLM-as-planner**), or formal programs including **PDDL** (1.2), **PDDL3**, **SMT** (more precisely, code to interact with the Python Z3 library), and **LTL**. The generation of all output types is done via one-shot prompting following cited work in Section 2. When generating formal programs, we consider three techniques: **Generation**, where the model directly generates the code in that formal language, **editing**, where the model first generates the code without considering the constraint, suggests an edit to that code to address the constraint, and re-generates the code accordingly, and **revision**, where the model is given up to 3 tries to revise the code via re-generation based on any error message returned by executing the code, if any.

Following the majority of cited work in Section 2, the predicted plan resulting from any method is validated against the ground-truth PDDL provided by `CoPE`, instead of being compared against "ground-truth" plans (Lyu et al., 2023; Liu et al., 2023b; Pan et al., 2023) since there could be multiple correct plans. We evaluate the predicted plans using the *correctness* metric which indicates the percentage that are successfully validated. For PDDL output, we use the `dual-bfws-ffparser` planner implemented by Muise (2016) as the solver and VAL (Howey et al., 2004) as the validator. For PDDL3, the output is first compiled using TCORE (Bonassi et al., 2021) into standard PDDL, then solved and validated the same way as PDDL. For SMT output, the LLM is responsible for generating code that includes a call to invoke the Z3 solver. For LTL, the LLM generates formulas representing environment dynamics, initial and goal states, and constraints. A satisfying plan is then produced by searching for a Lasso-shaped accepting run over the conjunction of all formulas. See more details in Appendix D. We use Deepseek-R1, Deepseek-V3 (Guo et al., 2025), Qwen3-32B (Team, 2025), and Qwen2.5-Coder-32B-Instruct (Hui et al., 2024) as LLMs that span different sizes of the ability to scale reasoning tokens during inference. We consciously only consider open-source models in light of open science.[2] We query Deepseek models with their API and Qwen models using KANI (Zhu et al., 2023) with default temperature on 1 H100 GPU. Prompts can be found in Appendix C.

6 Results

With evaluation on `CoPE`, we answer an array of research questions, draw findings, and provide actionable recommendations for future work.

How challenging are the constraints? Figure 2 and 6 display the performance of all methods, lan-

[2] See requirement H14 in `https://aclrollingreview.org/reviewerguidelines`.

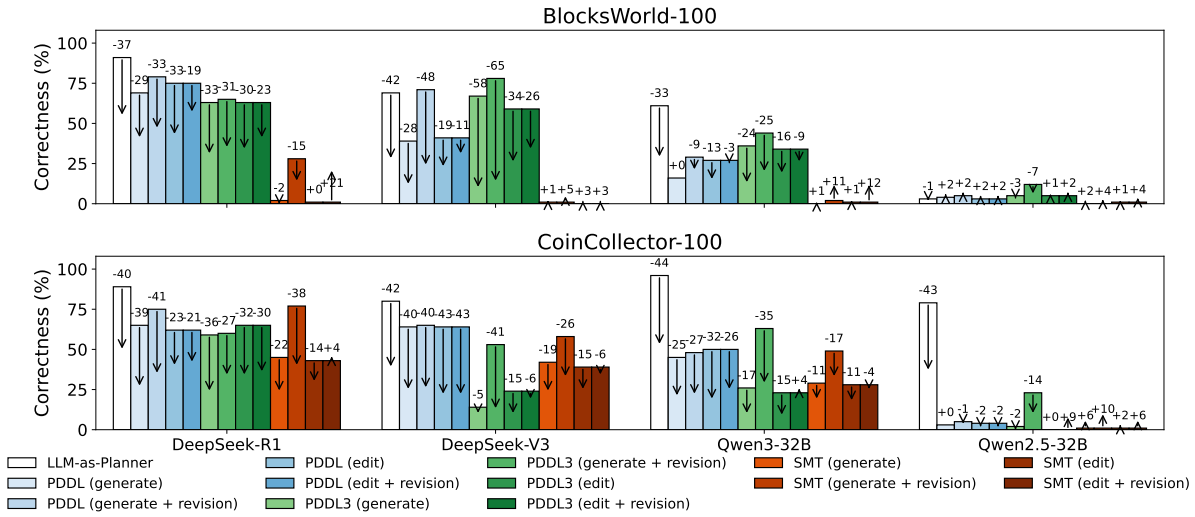

Figure 2: Performance of combinations of LLMs and methods on two domains. The arrows represent the performance difference once constraints are added. Revision is not included in this result to demonstrate that without extra help, this task is difficult. For all results, see Appendix E.

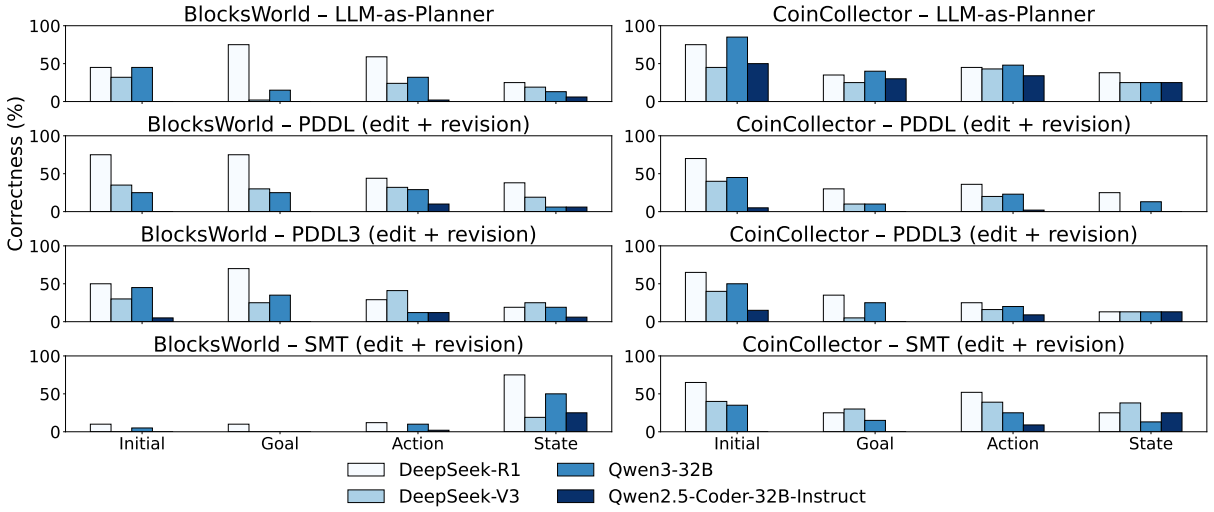

Figure 3: Best Performance of LLM-as-planner and LLM-as-formalizer (Generate) and (Edit) by constraint categories on `BlocksWorld-100` and `CoinCollector-100`. Revision improved LLM-as-formalizer for all methods and SMT performed better than PDDL on `CoinCollector-100`.

guages, techniques, LLMs, and domains. It is evident that simple one-line **constraints greatly diminish performance in both planning and formalizing**. While all experiments maintain non-trivial correctness greater than 20%, introducing constraints consistently degrades (often halves) the planning performances over all cases. Overall, LLM-as-planner outperforms its LLM-as-formalizer counterparts before and after the introduction of constraints. However, LLM-as-formalizer, with interpretability and formal guarantee, shows competitive performance with larger DeepSeek models and using revision by solver errors. With constraints, it benefits from the two-step generate-then-edit technique over simply generating PDDL in some but not all cases. Appendix Figure 7 shows that once constraints are introduced, the number of syntax errors increases for most methods. Comparing the two domains, CoinCollector (2 actions) is much simpler than BlocksWorld (4 actions) with regard to LLM-as-planner and LLM-as-SMT-formalizer but comparable with regard to other methods. Intuitively, this evidences LLM-as-PDDL-formalizer's robustness against the complexity by the number of actions over other methods. Regarding the choice of formal language, PDDL

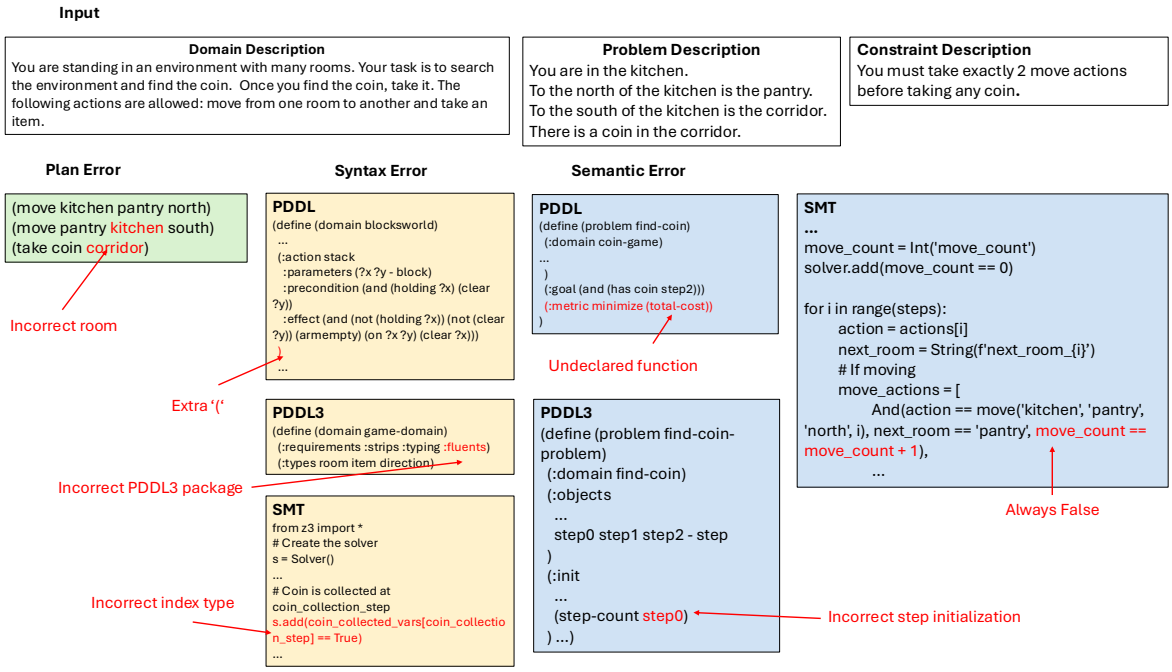

Figure 4: Examples of different errors found in model outputs.

greatly outperforms SMT in most cases, likely because PDDL more closely expresses the planning problem. However, while seemingly a more appropriate language which models constraints, PDDL3 generation is consistently worse than PDDL due to an elevated amount of compilation and syntax errors, likely because PDDL3 is even lower-resource for LLM pre-training.

When do models fail? Figure 4 qualitatively demonstrates an example set of errors for a particular constraint. A single constraint may cause various types of errors. For LLM-as-planner this set of errors includes hallucinating plans that do not exist, not satisfying newly added constraints and not following preconditions or effects set by the actions in the domain. The set of errors for LLM-as-formalizer includes syntax errors of PDDL, PDDL3 or Python, declaration of logically-incorrect functions, predicates, action parameters, preconditions in PDDL and PDDL3, or falsely encoding of constraints in SMT.

What category of constraints is hard? One may hypothesize that some categories of constraints may pose more challenge than others for a particular method. Recalling the definitions in Section 3, local edits may allow LLMs as PDDL formalizers to satisfy initial and goal constraints more easily compared to state constraints, since they do not require introducing and applying new predicates. In contrast, state constraints may be much easier to model using SMT which provides analogous functions. Moreover, PDDL3 may do well on both initial and goal constraints (due to local edits), as well as state constraints since the syntax can now encode constraints. Figure 3 show some yet non-decisive evidence towards those hypotheses. Appendix Figure 6 demonstrates that state constraints are more difficult than action constraints for LTL using DeepSeek models. Furthermore, PDDL3 demonstrated similar results to standard PDDL and in contrast to initial assumptions, performed the worst on state constraints. The most common errors include syntax errors when compiling PDDL3 into PDDL and semantic errors when finding a plan after compilation.

Can LLMs scale with complexity? Recent work has cast doubt on the ability of LLMs, as planner and formalizers, to scale with the problem complexity (Shojaee et al., 2025; Lin et al., 2025; Amonkar et al., 2025). In domains considered in this work, problems from `BlocksWorld-100` range from 2 to 15 blocks, while those from `CoinCollector-100` range from 3 to 11 rooms. Although our results represented in Figure 2 pose a significant room for improvement in model performance, especially after the introduction of constraints, it would only be more meaningful to conduct studies on problems with higher complexity emulate more

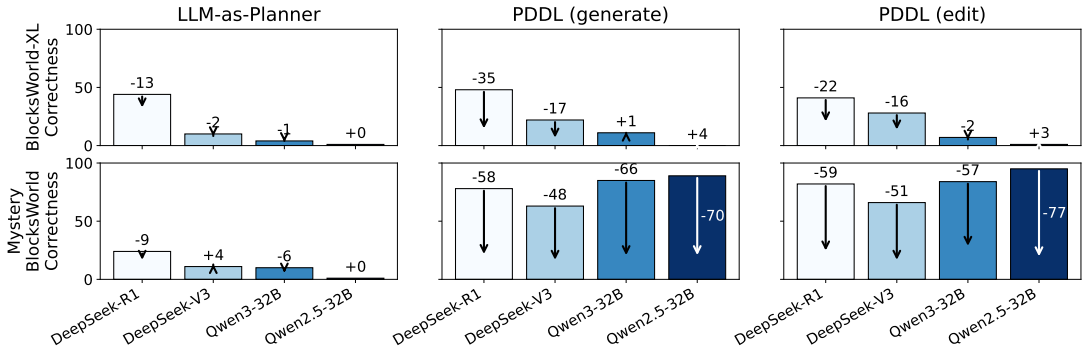

Figure 5: Performance of various methods on `BlocksWorld-XL-100` and `MysteryBlocksWorld-100` data to study models' robustness to complexity and lexical perturbation respectively, under the influence of constraints. As before, the arrows represent the performance difference once constraints are added. Results for SMT are omitted due to near-zero performance on both datasets.

to real life applications. Therefore, we present `BlocksWorld-XL-100`, a dataset included in CoPE where all problems contain 50 blocks, in order to see the impact of problems with a larger entity space on planning, especially with constraints.

Comparing performance of the first row of Figure 5 to that on `BlocksWorld-100` (Figure 2), performance for all methods sharply decreases as the problems become more complex even without the introduction of constraints, in line with cited literature. Even so, LLM-as-formalizer displays more robustness to complexity than LLM-as-planner. However, **such scaling behavior disappears with the constraints** as performance degrades by more than 50% for the well-performing DeepSeek models. This finding clearly indicates the challenge caused by the constraints aggravates the existing concern of scaling inability in LLM-assisted planning methodologies.

Do LLMs memorize the domains? Among the domains included in CoPE, BlocksWorld has existed for decades; and the simple semantics of navigation in CoinCollector have existed in many longstanding planning domains. As state-of-the-art LLMs are trained with most publicly available data, data contamination has been a stubborn challenge that hinders research (Sainz et al., 2023). As we have discussed in Section 2, we pose CoPE and the introduction of constraints as a remedy to shift the semantics in the data and bring models out of their comfort zone. We take one step further and apply the lexical obfuscation method of Mystery BlocksWorld (Valmeekam et al., 2024a) to `BlocksWorld-100` to construct `MysteryBlocksWorld-100`, by replacing all the names of the types, predicates, actions, and objects with nonsensical placeholders. We add our constraints by similarly swapping out the concepts in the constraints and the ground-truth PDDL.

From the second row of Figure 5, the performance without constraints is consistent with that of Huang and Zhang (2025), where LLM-as-planner is devastated compared to the original `BlocksWorld-100`, suggesting some extent of memorization, while LLM-as-formalizer remains robust to lexical perturbation. However, **such robustness disappears with the constraints**, as the performance for all 4 LLMs and 2 formalizing methods decreases by two thirds or more.

Our results jointly show that LLM-based planning and formalization are far from achieving robust performance on complex problems in niche domains. They also emphasize the importance of future benchmarks like CoPE that transcend simple problems in typical domains.

7 Conclusion

We introduce CoPE, a benchmark that augments existing planning domains with linguistically rich and formally categorized constraints to increase complexity and real-world applicability. Results show that CoPE presents significant challenges and consistently degrades performance. We provide a detailed analysis of behavioral differences of methods and planning languages, suggesting future research on tooling for LLM-based planning. Importantly, our constraints amplify models' lack of robustness with increasing problem complexity and lexical perturbation, calling for attention to the gap between benchmarks and real-world planning.

8 Limitation

Despite our best effort in defining, formalizing, and categorizing the constraints, we inevitably fall short of the goal of addressing all possible constraints (or more generally, requests) expressed in natural language. Even within our formalism, interesting phenomena such as the conjunction, negation, or ambiguity of the constraints are left for future work.

Our evaluation metric, plan correctness, may induce false positives where the plan may happen to be correct, but the generated code does not actually correctly describe the environment satisfy the constraint, leading to concerns in interpretability and faithfulness. However, to the best of our knowledge there is no feasible alternative. While there exists methods to evaluate generated PDDL problem files (Zuo et al., 2024) or domain files (Coulter et al., 2022) against a ground-truth, fundamentally there exists non-unique ways to formulate the same constraint.

While `CoPE` augments 4 datasets (all under the MIT License) in the BlocksWorld and CoinCollector domains, they are still oversimplified proof of concept and not representative to problems in the real world. This may pose a risk to users using this code on real world problems. Still, the challenge to state-of-the-method as we have demonstrated argues for the necessity of incremental evaluation.

References

Rikhil Amonkar, May Lai, Ronan Le Bras, and Li Zhang. 2025. Are llms better formalizers than solvers on complex problems? *Preprint*, arXiv:2505.13252.

Ujjwala Anantheswaran, Himanshu Gupta, Kevin Scaria, Shreyas Verma, Chitta Baral, and Swaroop Mishra. 2024. Cutting through the noise: Boosting llm performance on math word problems.

Luigi Bonassi, Alfonso Emilio Gerevini, Francesco Percassi, and Enrico Scala. 2021. On planning with qualitative state-trajectory constraints in pddl3 by compiling them away. *Proceedings of the International Conference on Automated Planning and Scheduling*, 31(1):46–50.

Alex Coulter, Teo Ilie, Renee Tibando, and Christian Muise. 2022. Theory alignment via a classical encoding of regular bisimulation. In *Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. 2022. From Spot 2.0 to Spot 2.10: What's new? In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)*, volume 13372 of *Lecture Notes in Computer Science*, pages 174–187. Springer.

Stefan Edelkamp. 2006. On the compilation of plan constraints and preferences. In *Proceedings of the Sixteenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'06, page 374–377. AAAI Press.

Richard E. Fikes and Nils J. Nilsson. 1971. Strips: a new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, IJCAI'71, page 608–620, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

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.

Alfonso Gerevini and Derek Long. 2005. Plan constraints and preferences in pddl3. Technical report, Technical Report 2005-08-07, Department of Electronics for Automation ....

Liancheng Gong, Wang Zhu, Jesse Thomason, and Li Zhang. 2025. Zero-shot iterative formalization and planning in partially observable environments. *Preprint*, arXiv:2505.13126.

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*.

Weihang Guo, Zachary Kingston, and Lydia E. Kavraki. 2024. Castl: Constraints as specifications through llm translation for long-horizon task and motion planning. *Preprint*, arXiv:2410.22225.

Yilun Hao, Yang Zhang, and Chuchu Fan. 2025. Planning anything with rigor: General-purpose zero-shot planning with LLM-based formalized programming. In *The Thirteenth International Conference on Learning Representations*.

Xingyun Hong, Yan Shao, Zhilin Wang, Manni Duan, and Xiongnan Jin. 2024. Towards building a robust knowledge intensive question answering model with large language models. *ArXiv*, abs/2409.05385.

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.

Mengkang Hu, Tianxing Chen, Yude Zou, Yuheng Lei, Qiguang Chen, Ming Li, Yao Mu, Hongyuan Zhang, Wenqi Shao, and Ping Luo. 2025. Text2world: Benchmarking large language models for symbolic world model generation. *Preprint*, arXiv:2502.13092.

Cassie Huang and Li Zhang. 2025. On the limit of language models as planning formalizers. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4880–4904, Vienna, Austria. Association for Computational Linguistics.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

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

Adam Ishay and Joohyung Lee. 2025. Llm+al: Bridging large language models and action languages for complex reasoning about actions. *Preprint*, arXiv:2501.00830.

Peter A. Jansen and Marc-Alexandre Côté. 2022. Textworldexpress: Simulating text games at one million steps per second. *arXiv*.

Prabhu Prakash Kagitha, Andrew Zhu, and Li Zhang. 2025. Addressing the challenges of planning language generation. *Preprint*, arXiv:2505.14763.

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*.

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. In *NeurIPS 2024*.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. 2025. Zebralogic: On the scaling limits of LLMs for logical reasoning. In *Forty-second International Conference on Machine Learning*.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. 2023. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. 2023. Clin: A continually learning language agent for rapid task adaptation and generalization. *Preprint*, arXiv:2310.10134.

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.

Mihir Parmar, Xin Liu, Palash Goyal, Yanfei Chen, Long Le, Swaroop Mishra, Hossein Mobahi, Jindong Gu, Zifeng Wang, Hootan Nakhost, Chitta Baral, Chen-Yu Lee, Tomas Pfister, and Hamid Palangi. 2025. Plangen: A multi-agent framework for generating planning and reasoning trajectories for complex problem solving. *Preprint*, arXiv:2502.16111.

Oscar Sainz, Jon Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. 2023. NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10776–10787, Singapore. Association for Computational Linguistics.

Keisuke Shirai, Cristian C. Beltran-Hernandez, Masashi Hamaya, Atsushi Hashimoto, Shohei Tanaka, Kento Kawaharazuka, Kazutoshi Tanaka, Yoshitaka Ushiku, and Shinsuke Mori. 2024. Vision-language interpreter for robot task planning. *Preprint*, arXiv:2311.00967.

Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. 2025. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *Preprint*, arXiv:2506.06941.

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

Katharina Stein, Daniel Fišer, Jörg Hoffmann, and Alexander Koller. 2025. Automating the generation of prompts for llm-based action choice in pddl planning. *Preprint*, arXiv:2311.09830.

Jing Han Sun and Ali Emami. 2024. Evograd: A dynamic take on the winograd schema challenge with human adversaries. In *International Conference on Language Resources and Evaluation*.

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.

Marcus Tantakoun, Christian Muise, and Xiaodan Zhu. 2025. LLMs as planning formalizers: A survey for leveraging large language models to construct automated planning models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 25167–25188, Vienna, Austria. Association for Computational Linguistics.

Qwen Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

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

Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. 2024b. Llms still can't plan; can lrms? a preliminary evaluation of openai's o1 on planbench. *Preprint*, arXiv:2409.13373.

Ruoyao Wang, Graham Todd, Eric Yuan, Ziang Xiao, Marc-Alexandre Côté, and Peter Jansen. 2023. Bytesized32: A corpus and challenge task for generating task-specific world models expressed as text games. *Preprint*, arXiv:2305.14879.

Hui Wei, Zihao Zhang, Shenghua He, Tian Xia, Shijia Pan, and Fei Liu. 2025. PlanGenLLMs: A modern survey of LLM planning capabilities. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 19497–19521, Vienna, Austria. Association for Computational Linguistics.

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*.

Ziyi Yang, Shreyas S. Raman, Ankit Shah, and Stefanie Tellex. 2023. Plug in the safety chip: Enforcing constraints for llm-driven robot agents. *Preprint*, arXiv:2309.09919.

Jianzhu Yao, Kevin Wang, Ryan Hsieh, Haisu Zhou, Tianqing Zou, Zerui Cheng, Zhangyang Wang, and Pramod Viswanath. 2025. Spin-bench: How well do llms plan strategically and reason socially? *Preprint*, arXiv:2503.12349.

Xingdi Yuan, Marc-Alexandre Côté, Alessandro Sordoni, Romain Laroche, Remi Tachet des Combes, Matthew Hausknecht, and Adam Trischler. 2019. Counting to explore and generalize in text-based games. *Preprint*, arXiv:1806.11525.

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.

Tianyi Zhang, Li Zhang, Zhaoyi Hou, Ziyu Wang, Yuling Gu, Peter Clark, Chris Callison-Burch, and Niket Tandon. 2024b. 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.

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, and Denny Zhou. 2024. Natural plan: Benchmarking llms on natural language planning.

Yinan Zheng, Ruiming Liang, Kexin ZHENG, Jinliang Zheng, Liyuan Mao, Jianxiong Li, Weihao Gu, Rui Ai, Shengbo Eben Li, Xianyuan Zhan, and Jingjing Liu. 2025. Diffusion-based planning for autonomous driving with flexible guidance. In *The Thirteenth International Conference on Learning Representations*.

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 Benchmark Examples

Listings 1 and 2 display a benchmark example of BlocksWorld from Valmeekam et al. (2024a) and (Huang and Zhang, 2025). Listings 3 and 4 display an example of the Logistics domain from Huang and Zhang (2025). Language seen in this benchmark is simplistic and not not realistic to the real world.

B Data Examples

We augment constraint descriptions to a ground-truth domain and problem file and create new ground-truth PDDL files that now encode the ground-truth. Here we include an example constraint description and the ground-truth PDDL files for the constraint + problem pair. Listings 5, 6 and 7 display an example constraint description and the annotated ground-truth $\mathcal{DF}$ and $\mathcal{PF}$ for a paired BlocksWorld problem. Listings 8, 9 and 10 display an example Domain, Problem and Constraint Description for `CoinCollector-100` as well as their groundtruth $\mathcal{DF}$ and $\mathcal{PF}$ in PDDL. Listing 11 shows an example ground-truth $\mathcal{PF}$ for the XL dataset, the $\mathcal{DF}$ remains the same. Listings 12, 13 and 14 display the corresponding constraint description, $\mathcal{DF}$ and $\mathcal{PF}$ that have now been obfuscated.

C Prompts

Listing 15 is the prompt for LLM-as-Planner. Listings 16, 17, 18 and 19 display the prompts for all experiment settings given to all models for LLM-as-PDDL-Formalizer. Listings 20, 21, 22 and 23 display the prompts for all experiment settings given to all models for LLM-as-PDDL3-Formalizer. Listings 24, 25 and 26 display prompts for experiments using SMT. Listing 27 displays all prompts for LTL. For PDDL and SMT, we asked the model to return the output in a JSON object whenever possible for easier parsing.

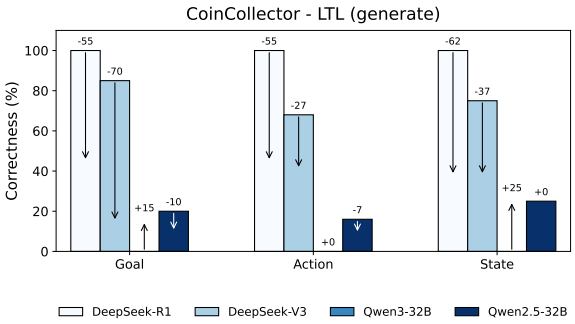

Figure 6: Performance of LTL on a subset of `CoinCollector-100` problems. The arrows represent the performance change with constraints. Details on design choice are discussed in Appendix D.3.

D Experimental Setup Details

D.1 Planner

We use a `dual-bfws-ffparser` planner to find plans from the generated PDDL. This is a best first width search planner used with a fast forward planner.

D.2 VAL

The VAL library takes in a ground-truth PDDL $\mathcal{DF}$, $\mathcal{PF}$ and a plan and tries to execute the plan in the environment by checking whether each action in the found plan can be executed based on the preconditions written in the ground-truth files. If the plan is executable, VAL then checks whether the final state after executing the plan matches the goal state found in the ground-truth $\mathcal{PF}$. If either the plan is not executable or the final state does not match the goal state, VAL will return an error, therefore the plan found by the planner is not correct.

D.3 LTL

Given problem description, the language model is asked to generate LTL formulas for the initial state, goal state, environment dynamics, and action dynamics. All formulas are concatenated into a conjunctive form as the formal representation for planning. When a constraint are introduced, we make a separate call to the language model for the formula representing the constraint and concatenate the formula to the existing conjunctive formula. We employ the spot library (Duret-Lutz et al., 2022) for encoding LTL formulas into Büchi automatons. Then, the state traces satisfying the problem specification can be found by searching for a Lasso-shaped accepting run. To make the results compatible with our evaluation pipeline, we

		Domains Used
LLM-as-formalizer	Huang and Zhang (2025)	BlocksWorld, Logistics, Barman
	Kagitha et al. (2025)	BlocksWorld, Logistics, Barman
	Gong et al. (2025)	CoinCollector, ALFWorld
	Hu et al. (2025)	Over 100 domains including Grid and BlocksWorld
	Li et al. (2024)	BEHAVIOR, VirtualHome
	Zuo et al. (2024)	BlocksWorld, Gripper
	Zhu et al. (2024)	Barman, BlocksWorld, Floortile, Grippers, Storage, Termes, TyreWorld
	Zhang et al. (2024a)	CoinCollector, CookingWorld
	Zhang et al. (2024b)	Proc2PDDL
	Liu et al. (2023a)	Barman, BlocksWorld, Floortile, Grippers, Storage, Termes, TyreWorld
	Xie et al. (2023)	BlocksWorld, ALFRED-L
	Wong et al. (2023)	Mini Minecraft, ALFRED
	Guan et al. (2023)	Household, Logistics, TyreWorld
	Shirai et al. (2024)	Cooking, BlocksWorld, Tower of Hanoi
	Ishay and Lee (2025)	MCP
	Hao et al. (2025)	Coffee, Workforce, Facility, Task Allocation, Warehouse, BlocksWorld, Mystery BlocksWorld, Movie, Gripper
	Guo et al. (2024)	HouseChip, Kitchen, BlocksWorld
	Tang et al. (2024)	Sokoban, Minigrid, ALFWorld
	Wang et al. (2023)	ByteSized32
	Amonkar et al. (2025)	NaturalPlan, ZebraLogic
	Lyu et al. (2023)	SayCan
	Gao et al. (2023)	GSM8K
LLM-as-planner	Shojaee et al. (2025)	Tower of Hanoi, Checker Jumping, BlocksWorld, River Crossing
	Parmar et al. (2025)	NaturalPlan, GPQA, OlympiadBench
	Majumder et al. (2023)	ScienceWorld
	Lin et al. (2023)	ScienceWorld
	Stein et al. (2025)	BlocksWorld, Depot, Logistics, Gripper, Ferry, Floortile, Goldminer, Grid, Gripper, Movie, Rovers, Satellite, Visitall
	Zheng et al. (2024)	NaturalPlan
	Stechly et al. (2024)	BlocksWorld
	Valmeekam et al. (2024a)	BlocksWorld, Logistics, Mystery BlocksWorld
	Yao et al. (2025)	BlocksWorld, Gripper, Barman, Logistics and more

Table 2: Comparison with related work.

map the traces back to sequences of PDDL actions by inferring actions from abstract state changes. In this process, the language model is also asked to generate environmental information, which is the room connectivities in CoinCollector domain.

We prompt the LLM to generate formulas for goal, action, and state constraints on `CoinCollector-100` only, since overriding initial states by editing is practically infeasible when all components of the environment and the problem are represented uniformly in a conjunctive formula. Models exhibit a similar performance degradation between 30-50% just as PDDL and SMT (Figure 6). Notably, different from PDDL and SMT, LTL is restricted to domains with simple action dynamics, e.g., navigational domains like CoinCollector, so while it performs well without constraints, it is not as generalizable as other languages.

E Full Results

Tables 3, 4, 5, 6 and 7 display results for all methodologies on `BlocksWorld-100` and `CoinCollector-100`. Tables 8 and 9 display results for `BlocksWorld-XL-100` and `MysteryBlocksWorld-100` on PDDL.

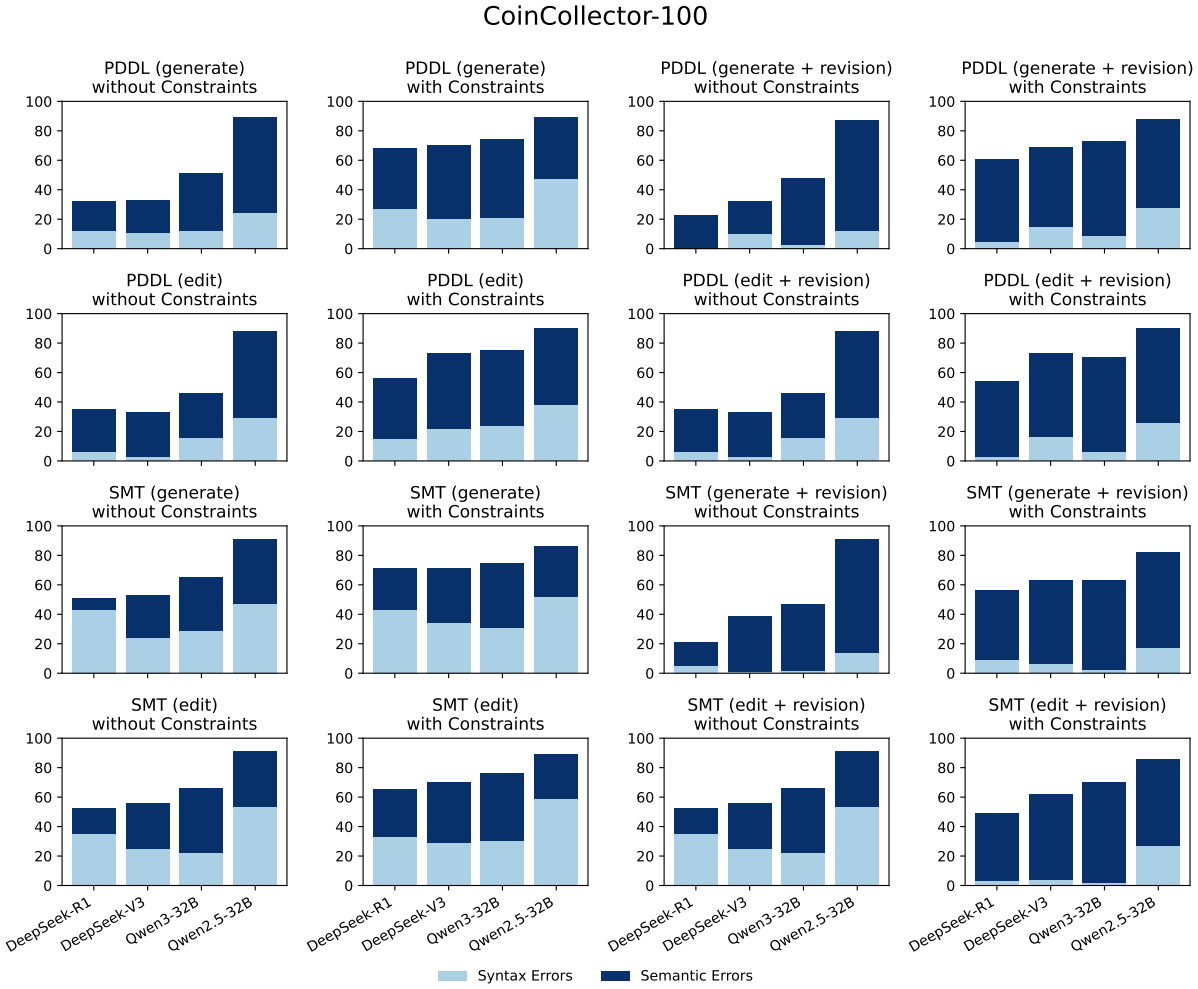

Figure 7: Counts of Syntax vs. Semantic Errors with and without Constraints on `CoinCollector-100` using both PDDL and SMT.

Listing 1: Domain Description for Moderately Templated BlocksWorld-100

```
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.
```

Listing 2: Problem Description for Moderately Templated BlocksWorld-100

```
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.
```

Listing 3: Domain Description for Moderately Templated Logistics-100

```
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.
```

Listing 4: Problem Description for Moderately Templated Logistics-100

```
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.
```

Listing 5: Example constraint description for an action constraint

```
Do not stack block1 on top of block2
```

Listing 6: Annotated ground-truth Domain File for BlocksWorld-100 with action constraint description

```
(define (domain blocksworld)
;; CONSTRAINT Do not stack block1 on top of block2.
  (:requirements :strips)
(:predicates (clear ?x)
            (on-table ?x)
            (arm-empty)
            (holding ?x)
            (on ?x ?y)
;; BEGIN ADD
             (do-not-stack ?x ?y)
;; END ADD
            )

(: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)
;; BEGIN ADD
                     (not (do-not-stack ?ob ?underob))
;; END ADD
                )
  :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)))))
```

Listing 7: Annotated ground-truth Problem File for BlocksWorld-100 with action constraint description

```
(define (problem blocksworld-p50)
;; CONSTRAINT Do not stack block1 on top of block2.
  (:domain blocksworld)
  (:objects block1 block2 block3 block4 block5 block6 block7 )
  (:init
    (on-table block3)
    (clear block3)
    (on-table block5)
    (clear block5)
    (on-table block7)
    (on block1 block7)
    (clear block1)
    (on-table block2)
    (clear block2)
    (on-table block6)
    (clear block6)
```

```
    (on-table block4)
    (clear block4)
    (arm-empty)
;; BEGIN ADD
    (do-not-stack block1 block2)
;; END ADD
  )
  (:goal (and
    (on-table block5)
    (on block2 block5)
    (on-table block4)
    (on block7 block4)
    (on block6 block7)
    (on block1 block6)
    (on-table block3)
  ))
)
```

Listing 8: Example Domain, Problem and Action Constraint Description for CoinCollector-100

```
;; Domain Description
You are standing in an environment with many rooms. Your task is to search the environment and find
     the coin. Once you find the coin, take it.

The following actions are allowed: move from one room to another and take an item.

;; Problem Description
You are in the kitchen.
To the east of the kitchen is the pantry.
To the west of the kitchen is the corridor.
There is a coin in the pantry.

;; Constraint Description
You must visit the corridor before the pantry.
```

Listing 9: Annotated ground-truth Domain File for CoinCollector problem with action constraint

```
(define (domain coin-collector)
;; CONSTRAINT You must visit the corridor before the pantry.
  (:requirements :strips :typing)
  (:types
    room
    direction
    item
  )
  (:predicates
    (at ?room - room)
    (connected ?room1 - room ?room2 - room ?direction - direction)
    (closed-door ?room1 - room ?room2 - room ?direction - direction)
    (location ?item - item ?room - room)
    (taken ?item - item)
    (is-reverse ?direction - direction ?reverse - direction)
;; BEGIN ADD
    (must-visit-before-next ?room - room)
    (next-room ?room - room)
    (visited-room)
;; END ADD
  )

  (:action move
    :parameters (?room1 - room ?room2 - room ?direction - direction)
    :precondition (and (at ?room1) (connected ?room1 ?room2 ?direction)
;; BEGIN ADD
    (not (and (next-room ?room2) (not (visited-room))))
;; END ADD
    )
    :effect (and (not (at ?room1)) (at ?room2)
;; BEGIN ADD
```

```
    (when (must-visit-before-next ?room2) (visited-room))
;; END ADD
    )
  )

  (:action take
    :parameters (?item - item ?room - room)
    :precondition (and (at ?room) (location ?item ?room) (not (taken ?item)))
    :effect (and (taken ?item) (not (location ?item ?room)))
  )
)
```

Listing 10: Annotated ground-truth Problem File for CoinCollector problem with action constraint

```
(define (problem coin_collector_numLocations3_numDistractorItems0_seed26)
;; CONSTRAINT You must visit the corridor before the pantry.
  (:domain coin-collector)
  (:objects
    kitchen pantry corridor - room
    north south east west - direction
    coin - item
  )
  (:init
    (at kitchen)
    (connected kitchen pantry east)
    (connected kitchen corridor west)
    (connected pantry kitchen west)
    (connected corridor kitchen east)
    (location coin pantry)
    (is-reverse north south)
    (is-reverse south north)
    (is-reverse east west)
    (is-reverse west east)
;; BEGIN ADD
    (must-visit-before-next corridor)
    (next-room pantry)
;; END ADD
  )
  (:goal
    (taken coin)
  )
)
```

Listing 11: Annotated ground-truth Problem File for the XL BlocksWorld-100 problems with action constraint description

```
(define (problem blocksworld-p50)
;; CONSTRAINT Do not stack block1 on top of block2.
  (:domain blocksworld)
  (:objects block1 block2 block3 block4 block5 block6 block7 block8 block9 block10 block11 block12
      block13 block14 block15 block16 block17 block18 block19 block20 block21 block22 block23
      block24 block25 block26 block27 block28 block29 block30 block31 block32 block33 block34
      block35 block36 block37 block38 block39 block40 block41 block42 block43 block44 block45
      block46 block47 block48 block49 block50 )
  (:init
    (on-table block10)
    (on block41 block10)
    (on block50 block41)
    (on block29 block50)
    (on block46 block29)
    (on block43 block46)
    (on block5 block43)
    (on block38 block5)
    (on block48 block38)
    (on block8 block48)
    (on block11 block8)
    (on block22 block11)
    (on block13 block22)
```

```
    (on block35 block13)
    (on block49 block35)
    (on block20 block49)
    (on block31 block20)
    (on block34 block31)
    (on block17 block34)
    (on block28 block17)
    (on block14 block28)
    (on block47 block14)
    (on block26 block47)
    (on block6 block26)
    (on block4 block6)
    (on block25 block4)
    (on block9 block25)
    (on block23 block9)
    (on block15 block23)
    (on block21 block15)
    (on block18 block21)
    (on block39 block18)
    (on block33 block39)
    (on block1 block33)
    (on block3 block1)
    (on block2 block3)
    (on block44 block2)
    (on block16 block44)
    (on block45 block16)
    (on block7 block45)
    (on block40 block7)
    (on block24 block40)
    (on block27 block24)
    (on block36 block27)
    (on block19 block36)
    (clear block19)
    (on-table block32)
    (clear block32)
    (on-table block42)
    (on block37 block42)
    (on block30 block37)
    (on block12 block30)
    (clear block12)
    (arm-empty)
;; BEGIN ADD
    (do-not-stack block1 block2)
;; END ADD
  )
  (:goal (and
    (on-table block46)
    (on block45 block46)
    (on block37 block45)
    (on-table block30)
    (on block39 block30)
    (on-table block34)
    (on-table block49)
    (on-table block19)
    (on-table block38)
    (on-table block16)
    (on-table block25)
    (on-table block18)
    (on block5 block18)
    (on block17 block5)
    (on-table block42)
    (on-table block3)
    (on block32 block3)
    (on block35 block32)
    (on block26 block35)
    (on-table block12)
    (on-table block33)
    (on-table block14)
    (on-table block2)
    (on-table block23)
```

```
    (on block1 block23)
    (on block6 block1)
    (on-table block4)
    (on block20 block4)
    (on-table block48)
    (on-table block27)
    (on-table block31)
    (on-table block44)
    (on block47 block44)
    (on-table block43)
    (on-table block36)
    (on-table block7)
    (on-table block22)
    (on-table block28)
    (on-table block9)
    (on-table block11)
    (on-table block50)
    (on-table block24)
    (on-table block41)
    (on-table block8)
    (on-table block21)
    (on-table block40)
    (on-table block13)
    (on-table block15)
    (on-table block29)
    (on-table block10)
  ))
)
```

Listing 12: Example constraint description for a action constraint for Mystery BlocksWorld

```
Do not perform action3 with object1 and object2.
```

Listing 13: Annotated ground-truth Domain File for Mystery BlocksWorld-100 with action constraint description

```
(define (domain mystery_blocksworld)
;; CONSTRAINT Do not perform action3 with object1 and object2.
  (:requirements :strips)
(:predicates (predicate1 ?x)
             (predicate2 ?x)
             (predicate3)
             (predicate4 ?x)
             (predicate5 ?x ?y)
;; BEGIN ADD
              (not-allowed ?x ?y)
;; END ADD
             )

(:action action1
  :parameters (?object1)
  :precondition (and (predicate1 ?object1) (predicate2 ?object1) (predicate3))
  :effect (and (predicate4 ?object1) (not (predicate1 ?object1)) (not (predicate2 ?object1))
               (not (predicate3))))

(:action action2
  :parameters (?object1)
  :precondition (predicate4 ?object1)
  :effect (and (predicate1 ?object1) (predicate3) (predicate2 ?object1)
               (not (predicate4 ?object1))))

(:action action3
  :parameters (?object1 ?object2)
  :precondition (and (predicate1 ?object2) (predicate4 ?object1)
;; BEGIN ADD
                     (not (not-allowed ?object1 ?object2))
;; END ADD
                 )
  :effect (and (predicate3) (predicate1 ?object1) (predicate5 ?object1 ?object2)
```

```
            (not (predicate1 ?object2)) (not (predicate4 ?object1))))

(:action action4
  :parameters (?object1 ?object2)
  :precondition (and (predicate5 ?object1 ?object2) (predicate1 ?object1) (predicate3))
  :effect (and (predicate4 ?object1) (predicate1 ?object2)
            (not (predicate5 ?object1 ?object2)) (not (predicate1 ?object1)) (not (predicate3)))))
```

Listing 14: Annotated ground-truth Problem File for Mystery BlocksWorld-100 with action constraint description

```
(define (problem mystery_blocksworld-p50)
;; CONSTRAINT Do not perform action3 with object1 and object2.
  (:domain mystery_blocksworld)
  (:objects object1 object2 object3 object4 object5 object6 object7 )
  (:init
    (predicate2 object3)
    (predicate1 object3)
    (predicate2 object5)
    (predicate1 object5)
    (predicate2 object7)
    (predicate5 object1 object7)
    (predicate1 object1)
    (predicate2 object2)
    (predicate1 object2)
    (predicate2 object6)
    (predicate1 object6)
    (predicate2 object4)
    (predicate1 object4)
    (predicate3)
;; BEGIN ADD
    (not-allowed object1 object2)
;; END ADD
  )
  (:goal (and
    (predicate2 object5)
    (predicate5 object2 object5)
    (predicate2 object4)
    (predicate5 object7 object4)
    (predicate5 object6 object7)
    (predicate5 object1 object6)
    (predicate2 object3)
  ))
)
```

Listing 15: Prompt for LLM-as-planner

```
You are a PDDL expert. Here is a game we are playing.
{domain_description}
{problem_description}
{constraint_description}

Write the plan that would solve this problem.

These are the available actions:
{available_actions}

Here is what the output should look like:
{example_answer}
```

Listing 16: Prompt for LLM-as-PDDL-formalizer, while generating the entire code

```
You are a PDDL expert. Here is a game we are playing.
{domain_description}
{problem_description}
{constraint_description}

Write the domain and problem files in minimal PDDL.
```

```
These are the available actions:
{available_actions}
```

Listing 17: Prompt for LLM-as-PDDL-formalizer, while first generating the non-constrained PDDL for the Edit setting

```
You are a PDDL expert. Here is a game we are playing.
{domain_description}
{problem_description}
Write the domain and problem files in minimal PDDL.

These are the available actions:
{available_actions}
```

Listing 18: Prompt for LLM-as-PDDL-formalizer, while modifying the non-constrained PDDL for the Edit setting

```
You are a PDDL expert. Here is a PDDL domain and problem file.
{original_domain_file}
{original_problem_file}

Modify the PDDL files so that it satisfies the following constraint: {constraint_description}
These are the available actions:
{available_actions}
```

Listing 19: Prompt for LLM-as-PDDL-formalizer, including revision

```
You are a PDDL expert. The following domain and problem files have the error: {error}

{domain_file}
{problem_file}

Revise the PDDL to remove the error.
```

Listing 20: Prompt for LLM-as-PDDL3-formalizer, while generating the entire code

```
You are a PDDL3 expert. Here is a game we are playing.
{domain_description}
{problem_description}
{constraint_description}

Write the domain and problem files in minimal PDDL3 code.

These are the available actions:
{available_actions}
```

Listing 21: Prompt for LLM-as-PDDL3-formalizer, while first generating the non-constrained PDDL for the Edit setting

```
You are a PDDL3 expert. Here is a game we are playing.
{domain_description}
{problem_description}
Write the domain and problem files in minimal PDDL3 code.

These are the available actions:
{available_actions}
```

Listing 22: Prompt for LLM-as-PDDL3-formalizer, while modifying the non-constrained PDDL for the Edit setting

```
You are a PDDL3 expert. Here is a PDDL3 domain and problem file.
{original_domain_file}
{original_problem_file}
```

```
Modify the files using PDDL3 so that it satisfies the following constraint: {constraint_description}
These are the available actions:
{available_actions}
```

Listing 23: Prompts for LLM-as-PDDL3-formalizer, including revision

```
;; Error when running Compiler
You are a PDDL3 expert. The following PDDL3 were compiled and resulted in an error. Revise the PDDL3
    to remove the error. Return a JSON object in the following format:
{{
\"domain file\": ...,
\"problem file\":...
}}

Domain File:
{original_domain_file}

Problem File:
{original_problem_file}

Error Message:
{compilation_error}

;; Error when running Solver
You are a PDDL3 expert. The following PDDL3 were compiled and ran through a solver, which resulted in
     an error. Revise the original PDDL3 to remove the error. Return a JSON object in the following
    format:
{{
\"domain file\": ...,
\"problem file\":...
}}

Original Domain File:
{original_domain_file}

Original Problem File:
{original_problem_file}

Compiled Domain File:
{compiled_domain_file}

Compiled Problem File:
{compiled_problem_file}

Error Message:
{solver_error}
```

Listing 24: Prompt for LLM-as-SMT-formalizer (generate)

```
You are a Z3 expert. Here is a domain and problem instance for {domain}.
{domain_description}
{problem_description}
{constraint_description}

Generate Python code that uses the Z3 Python API to solve this instance. These are the available
    actions:
{available_actions}

The output of your Python code should be a plan in the following format:{example_plan}

Set the number of steps allowed to 100.
```

Listing 25: Prompts for LLM-as-SMT-formalizer (edit)

```
;; Step 1
You are a Z3 expert. Here is a domain and problem instance for {domain}.
{domain_description}
```

```
{problem_description}
Generate Python code that uses the Z3 Python API to solve this instance. These are the available
    actions:
{available_actions}

The output of your Python code should be a plan in the following format:
{example_plan}

Set the number of steps allowed to 100.

;; Step 2
You are a Z3 expert. Here is a {domain} instance written in Z3.
{original_python_code}

Modify the Z3 code so that it satisfies the following constraint: {constraint_description}
These are the available actions:
{available_actions}

The output of your Python code should be a plan in the following format:
{example_plan}
Set the number of steps allowed to 100.
```

Listing 26: Prompt for revision step for LLM-as-SMT-formalizer

```
You are a Z3 expert. The following file have the error: {error}

{python_file}

Revise the Z3 code to remove the error.
```

Listing 27: Prompt for LLM-as-LTL-Formalizer

```
;; coin_collector_formula
  You are given a description of a grid world with rooms, adjacencies (north/south/east/west
      relations),
  and possibly items (e.g. a coin) located in some room.

  Your task is to convert this description into a set of LTL formulas for planning.
  Decompose the problem into 5 parts:
  1. Initial condition
  2. Occupancy (exactly one room at a time)
  3. Move dynamics (allowed moves)
  4. Pickup dynamics (pickup conditions)
  5. Goal (eventual condition)

  ### Rules:
  - Each room should be an atomic proposition in its name (e.g., `kitchen`, `bedroom`, etc.).
  - Exactly one room must be true at any time.
  - Legal moves: if you are in room A, the next state must be in one of the adjacent rooms or staying
       at the current room.
  - Legal pickups: if there is a coin in a room R, define an atomic proposition `has_coin`, and add
      these two rules to pickup dynamics for `has_coin`:
    - If the agent is in R and does not have the coin, then in the next state it may either gain the
        coin or stay without it.
    - If the agent is not in R and does not have the coin, then in the next state it must still not
        have the coin.
    - If the agent has the coin, it continues to have it in all future states.
    - if the coin is in room R, then picking it up must only change `has_coin` but not coincide with
        moving.
  - Initial condition: mark the starting room. `has_coin` is always false initially.
  - Goal: eventually `has_coin` must be true.

  ### Output format:
  Produce 4 formulas in Python strings:
  - `init = "..."`
  - `occupancy = "..."`
  - `move_dynamics = "..."`
  - `pickup_dynamics = "..."`
```

```
  - `goal = "..."`

  ---
  [DESCRIPTION]
  ---
  Now generate the formulas:

;; coin_collector_adjacency
  You are given a description of a world with rooms and directional relations.

  Your task is to output the adjacency dictionary in Python format.

  ### Rules:
  - Use room names exactly as given in the text, lowercased and underscores instead of spaces.
  - Each key in the dictionary is a room.
  - Each value is another dictionary: {neighbor: direction}.
  - The direction must be from the perspective of the key room (north, south, east, west).
  - Always include both directions if they are implied.
    Example: "To the north of the kitchen is the pantry" means:
        adjacency["kitchen"]["pantry"] = "north"
        adjacency["pantry"]["kitchen"] = "south"

  ### Output format:
  adjacency = {
  "room1": {"neighbor1": "direction", "neighbor2": "direction", ...},
  "room2": {...},
  ...
  }

  ---
  [DESCRIPTION]
  ---
  Now generate only the adjacency dictionary in the output format:

;; coin_collector_constraints
  You are given a set of atomic propositions and an environment description that represent a grid
      world planning problem.
  - Each room name is an atomic proposition, e.g.,[ROOMS]
  - The agent may pick up a coin, tracked by proposition "has_coin".
  Environment description:
  [PROBELM_DESCRIPTION]

  Your task: Given an additional natural language constraint, output the corresponding LTL formula
      that enforces it.

  ### Rules:
  - Always output formulas in the form of Python strings, e.g., "G(...)" or "F(...)".
  - Use the existing atomic propositions exactly as defined.
  - If the constraint mentions a room, use the room's name directly.
  - You should only output formulas for the new constraints.

  ### Examples:
  - If a constraint says "never enter ROOM", output `"G(!ROOM)"`.
  - If a constraint says "only move south", expand it into formulas of the form:
      For each room r, if its southern neighbor is s, then:
        G( r -> X(s) )
      and forbid all other neighbors in the X operator.
  - If a constraint says "only move south or west", expand similarly, listing all allowed successors
      explicitly.
  - If a constraint says "must visit ROOM_A before ROOM_B", encode it as `"(!ROOM_B) U ROOM_A"`.
  - If a constraint says "visit ROOM_A right after pick up coin", be careful about the number of X
      operators:
      If you are picking up the coin in the next timestep, visiting room a will happen in the
          following timestep: "G( (!has_coin & X(has_coin)) -> XX(ROOM_A) )"
  - Always output Python string formulas, one per constraint.
  - Do not introduce undefined macros.

  ### Formatting Examples:
  Natural language: "The agent must never enter the backyard."
  Output: "G(!backyard)"
```

```
Natural language: "The agent must visit the bathroom before visiting the bedroom."
Output: "(!bedroom) U bathroom"

---
Constraint Description:
[CONSTRAINTS_DESCRIPTION]
Now, given these natural language constraints, output only the corresponding LTL formulas, without
    explanations:
```

formalizer type	method	model	correctness (non-constrained)	correctness (constrained)
llm-as-planner	N/A	deepseek-reasoner	91	54
		deepseek-chat	69	27
		Qwen3-32B	61	28
		Qwen2.5-Coder-32B-Instruct	3	2
llm-as-pddl-formalizer	generate	deepseek-reasoner	69	40
		deepseek-chat	39	11
		Qwen3-32B	16	16
		Qwen2.5-Coder-32B-Instruct	4	6
	generate + revision	deepseek-reasoner	79	46
		deepseek-chat	71	23
		Qwen3-32B	29	20
		Qwen2.5-Coder-32B-Instruct	5	7
	edit	deepseek-reasoner	75	42
		deepseek-chat	41	22
		Qwen3-32B	27	14
		Qwen2.5-Coder-32B-Instruct	3	5
	edit + revision	deepseek-reasoner	75	56
		deepseek-chat	41	30
		Qwen3-32B	27	24
		Qwen2.5-Coder-32B-Instruct	3	5
llm-as-smt-formalizer	generate	deepseek-reasoner	2	0
		deepseek-chat	1	2
		Qwen3-32B	0	1
		Qwen2.5-Coder-32B-Instruct	0	2
	generate + revision	deepseek-reasoner	28	13
		deepseek-chat	1	6
		Qwen3-32B	2	13
		Qwen2.5-Coder-32B-Instruct	0	4
	edit	deepseek-reasoner	1	1
		deepseek-chat	0	3
		Qwen3-32B	1	2
		Qwen2.5-Coder-32B-Instruct	1	2
	edit + revision	deepseek-reasoner	1	22
		deepseek-chat	0	3
		Qwen3-32B	1	13
		Qwen2.5-Coder-32B-Instruct	1	5

Table 3: Correctness with and without constraints on `BlocksWorld-100` using all methods.

formalizer type	method	model	correctness (non-constrained)	correctness (constrained)
llm-as-planner	N/A	deepseek-reasoner	89	49
		deepseek-chat	80	38
		Qwen3-32B	96	52
		Qwen2.5-Coder-32B-Instruct	79	36
llm-as-pddl-formalizer	generate	deepseek-reasoner	65	26
		deepseek-chat	64	24
		Qwen3-32B	45	20
		Qwen2.5-Coder-32B-Instruct	3	3
	generate + revision	deepseek-reasoner	75	34
		deepseek-chat	65	25
		Qwen3-32B	48	21
		Qwen2.5-Coder-32B-Instruct	5	4
	edit	deepseek-reasoner	62	39
		deepseek-chat	64	21
		Qwen3-32B	50	18
		Qwen2.5-Coder-32B-Instruct	4	2
	edit + revision	deepseek-reasoner	62	41
		deepseek-chat	64	21
		Qwen3-32B	50	24
		Qwen2.5-Coder-32B-Instruct	4	2
llm-as-smt-formalizer	generate	deepseek-reasoner	45	23
		deepseek-chat	42	23
		Qwen3-32B	29	18
		Qwen2.5-Coder-32B-Instruct	1	7
	generate + revision	deepseek-reasoner	77	39
		deepseek-chat	58	32
		Qwen3-32B	49	32
		Qwen2.5-Coder-32B-Instruct	1	11
	edit	deepseek-reasoner	43	29
		deepseek-chat	39	24
		Qwen3-32B	28	17
		Qwen2.5-Coder-32B-Instruct	1	3
	edit + revision	deepseek-reasoner	43	47
		deepseek-chat	39	33
		Qwen3-32B	28	24
		Qwen2.5-Coder-32B-Instruct	1	7

Table 4: Correctness with and without constraints on `CoinCollector-100` using all LLM-as-Planner, LLM-as-PDDL-Formalizer, LLM-as-SMT-Formalizer.

formalizer type	method	model	correctness (non-constrained)	correctness (constrained)
llm-as-ltl-formalizer	generate	deepseek-reasoner	71	27
		deepseek-chat	50	18
		Qwen3-32B	0	15
		Qwen2.5-Coder-32B-Instruct	7	12

Table 5: Correctness with and without constraints on `CoinCollector-100` using LLM-as-LTL-Formalizer.

formalizer type	method	model	correctness (non-constrained)	correctness (constrained)
llm-as-pddl3-formalizer	generate	deepseek-reasoner	61	29
		deepseek-chat	64	9
		Qwen3-32B	35	12
		Qwen2.5-Coder-32B-Instruct	5	2
	generate + revision	deepseek-reasoner	63	33
		deepseek-chat	76	13
		Qwen3-32B	43	18
		Qwen2.5-Coder-32B-Instruct	12	5
	edit	deepseek-reasoner	61	32
		deepseek-chat	57	24
		Qwen3-32B	33	17
		Qwen2.5-Coder-32B-Instruct	5	6
	edit + revision	deepseek-reasoner	61	39
		deepseek-chat	57	32
		Qwen3-32B	33	24
		Qwen2.5-Coder-32B-Instruct	5	7

Table 6: Correctness with and without constraints on `BlocksWorld-100` using PDDL3.

formalizer type	method	model	correctness (non-constrained)	correctness (constrained)
llm-as-pddl3-formalizer	generate	deepseek-reasoner	54	21
		deepseek-chat	13	8
		Qwen3-32B	24	8
		Qwen2.5-Coder-32B-Instruct	2	0
	generate + revision	deepseek-reasoner	55	30
		deepseek-chat	49	11
		Qwen3-32B	58	26
		Qwen2.5-Coder-32B-Instruct	21	8
	edit	deepseek-reasoner	60	30
		deepseek-chat	22	8
		Qwen3-32B	21	7
		Qwen2.5-Coder-32B-Instruct	0	0
	edit + revision	deepseek-reasoner	60	32
		deepseek-chat	22	17
		Qwen3-32B	21	25
		Qwen2.5-Coder-32B-Instruct	0	8

Table 7: Correctness with and without constraints on `CoinCollector-100` using PDDL3.

formalizer type	method	model	correctness (non-constrained)	correctness (constrained)
llm-as-planner	N/A	deepseek-reasoner	43	30
		deepseek-chat	10	8
		Qwen3-32B	4	3
		Qwen2.5-Coder-32B-Instruct	1	1
llm-as-pddl-formalizer	generate	deepseek-reasoner	47	13
		deepseek-chat	21	5
		Qwen3-32B	11	12
		Qwen2.5-Coder-32B-Instruct	0	4
	generate + revision	deepseek-reasoner	49	18
		deepseek-chat	54	13
		Qwen3-32B	22	14
		Qwen2.5-Coder-32B-Instruct	0	7
	edit	deepseek-reasoner	40	18
		deepseek-chat	27	12
		Qwen3-32B	7	5
		Qwen2.5-Coder-32B-Instruct	1	4
	edit + revision	deepseek-reasoner	40	25
		deepseek-chat	27	22
		Qwen3-32B	7	15
		Qwen2.5-Coder-32B-Instruct	1	5

Table 8: Correctness with and without constraints on `BlocksWorld-XL-100` using PDDL.

formalizer type	method	model	correctness (non-constrained)	correctness (constrained)
llm-as-planner	N/A	deepseek-reasoner	23	15
		deepseek-chat	11	6
		Qwen3-32B	10	4
		Qwen2.5-Coder-32B-Instruct	1	1
llm-as-pddl-formalizer	generate	deepseek-reasoner	76	19
		deepseek-chat	61	15
		Qwen3-32B	82	18
		Qwen2.5-Coder-32B-Instruct	86	18
	generate + revision	deepseek-reasoner	95	24
		deepseek-chat	91	23
		Qwen3-32B	93	26
		Qwen2.5-Coder-32B-Instruct	86	18
	edit	deepseek-reasoner	80	22
		deepseek-chat	64	15
		Qwen3-32B	81	26
		Qwen2.5-Coder-32B-Instruct	92	17
	edit + revision	deepseek-reasoner	80	34
		deepseek-chat	64	23
		Qwen3-32B	81	27
		Qwen2.5-Coder-32B-Instruct	92	17

Table 9: Correctness with and without constraints on `MysteryBlocksWorld-100` using all PDDL.