Planetarium 🪐: A Rigorous Benchmark for Translating Text to Structured Planning Languages

Max Zuo [*]   **Francisco Piedrahita Velez** [*]   **Xiaochen Li**
Michael L. Littman   **Stephen H. Bach**
[1]Department of Computer Science, Brown University
{zuo, francisco, xiaochen_li, michael_littman, stephen_bach}@brown.edu

Abstract

Recent works have explored using language models for planning problems. One approach examines translating natural language descriptions of planning tasks into structured planning languages, such as the planning domain definition language (PDDL). Existing evaluation methods struggle to ensure semantic correctness and rely on simple or unrealistic datasets. To bridge this gap, we introduce *Planetarium*, a benchmark designed to evaluate language models' ability to generate PDDL code from natural language descriptions of planning tasks. *Planetarium* features a novel PDDL equivalence algorithm that flexibly evaluates the correctness of generated PDDL, along with a dataset of 145,918 text-to-PDDL pairs across 73 unique state combinations with varying levels of difficulty. Finally, we evaluate several API-access and open-weight language models that reveal this task's complexity. For example, 96.1% of the PDDL problem descriptions generated by GPT-4o are syntactically parseable, 94.4% are solvable, but only 24.8% are semantically correct, highlighting the need for a more rigorous benchmark for this problem.
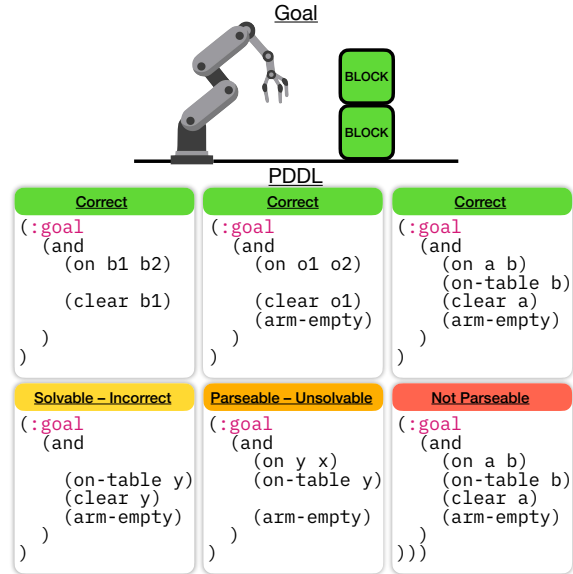
Figure 1: An example of one planning goal corresponding to many correct PDDL goals. All PDDL goals in the top row represent the displayed goal correctly. The bottom row illustrates PDDL goals with different error types, showing instances that are solvable (a planner can generate a plan, but for a different planning problem), parseable (the PDDL syntax is correct but will not produce any plan from a planner), and not parseable (it is not valid PDDL). See Section 5 for details.

1 Introduction

Recently, there has been growing interest in using large language models (LLMs) to solve planning problems. Some research has focused on generating plans directly with LLMs (Valmeekam et al., 2023a, 2022, 2023b; Silver et al., 2022, 2023). However, this approach has shown limited success; GPT-4 only achieves 35% accuracy on simple planning problems (Valmeekam et al., 2023a). Another line of research uses LLMs to convert natural language prompts into structured planning languages, such as the planning domain definition language (Liu et al., 2023; Xie et al., 2023; Guan et al., 2023; Chalvatzaki et al., 2023; Yang et al., 2023b). Early evidence suggests this method performs better than

generating plans directly with LLMs (Liu et al., 2023). Despite its promise, there are no rigorous techniques or benchmarks for evaluating the translation of natural language planning descriptions to PDDL.

Translating natural language to PDDL enables a hybrid, best-of-both-worlds approach. The LLM is responsible for interpreting natural language, and the resulting PDDL can be given to traditional, symbolic planners that have been developed over decades (Fikes and Nilsson, 1971; McDermott et al., 1998; Helmert, 2006; Baier et al., 2009; Vidal and Geffner, 2006). These traditional planners are efficient and ensure the correctness of their so-

---

[*]These authors contributed equally to this work.

lutions. In contrast, leaving the planning to LLMs does not guarantee correctness, making them unreliable for critical applications. Despite this shortcoming, LLMs are an enticing approach, as using traditional planners requires domain expertise and expertise in modeling planning problems. By using LLMs to translate natural language into PDDL, we can better leverage the strengths of existing planners.

Using LLMs to translate natural language to PDDL is an instance of a code generation task. Evaluating code generation tasks, in general, is highly challenging. Some benchmarks for code generation use match-based metrics that look for segments that overlap with ground truth code (Papineni et al., 2002; Lin, 2004; Ren et al., 2020), but match-based metrics cannot cover the vast space of equivalent programs (Chen et al., 2021). Therefore, benchmarks often test functional correctness using a suite of unit tests (Roziere et al., 2020; Kulal et al., 2019). For planning problems, existing work uses "plan validators" to check if the generated code can be solved with a traditional planner (Liu et al., 2023; Silver et al., 2022, 2023; Guan et al., 2023). We argue using validators alone is insufficient to determine if PDDL generation is correct. This is because the LLM can generate valid PDDL that has *nothing* to do with the user's instructions and still be considered correct—a false positive.

A rigorous evaluation of LLMs as generators of structured planning languages requires a precise definition of what it means for generated code to be correct. This is hard because many instances of PDDL can represent the same planning problem, but it is not always obvious when they are equivalent (Figure 1). Properly checking PDDL equivalence requires symbolic interpretation.

To address this challenge, we introduce *Planetarium*, a benchmark to evaluate LLMs on translating natural language descriptions of planning problems into PDDL. Our contributions are as follows:

Rigorous Evaluation of PDDL Equivalence. We formally define planning problem equivalence and create an algorithm for checking whether two PDDL problems satisfy this definition. This algorithm transforms the PDDL code into scene graphs, computes an expansion of the goal states for both PDDL problems, and then performs isomorphism checks between the graphs. Our method ensures two PDDL problems match if and only if they represent the same underlying planning task. We

show how to make this algorithm efficient for three domains: Blocks World, Gripper, and a slightly simplified Floor Tile (McDermott, 2000; Linares López et al., 2015).

Benchmark Data for PDDL Generation. We present a dataset based on the International Planning Competition (IPC) (McDermott, 2000; Vallati et al., 2015; Taitler et al., 2024; Seipp et al., 2022), crafting 145,918 ground truth PDDL problems and corresponding text descriptions capturing a range of planning problems. Each task varies in two dimensions to assess the difficulty of PDDL generation: abstraction and size.

Broad Evaluation of Current LLMs. Finally, we evaluate a range of API-access and open-weight LLMs on *Planetarium*. We evaluate in both a zero-shot setting and after fine-tuning. We find that this task is very challenging. GPT-4o in a zero-shot setting gets only 24.8% correct. Instances with abstract descriptions or many propositions are particularly challenging. *Planetarium* can, therefore, serve as a benchmark of progress on this important problem. To support future development and evaluation of LLMs, we release all the code and data for *Planetarium* at `github.com/BatsResearch/planetarium`.

2 Related Work

LLMs as Planners. There is growing interest in using LLMs to solve planning problems. Several papers have demonstrated the potential of using LLMs for decision-making (Sharma et al., 2022; Ren et al., 2023; Ichter et al., 2022; Singh et al., 2022). Some techniques that enable the use of LLMs as planners involve decomposing abstract tasks into concrete, atomic executable actions for an agent to perform (Huang et al., 2022b,a; Sharma et al., 2022). Other approaches generate plan actions by scoring the possible next steps in a sequence of actions (Ichter et al., 2022; Ren et al., 2023). When using LLMs to generate plans directly, the LLM is given a natural language representation of a planning problem with the goal of generating a plan. We refer to this line of work as "*LLMs as planners*." One of the main findings is that LLMs have limited ability to generate and validate plans on their own, even for simple planning tasks (Valmeekam et al., 2022; Silver et al., 2022; Valmeekam et al., 2023b).

Planner-Augmented LLMs. Given LLMs' poor performance in classical planning tasks (Valmeekam et al., 2023a, 2022; Silver et al., 2022), new approaches extend LLMs by incorporating classical planning techniques (Liu et al., 2023; Guan et al., 2023). Some techniques frame the problem as a machine translation task, transforming a natural language text description of a planning problem into PDDL to use with classical planners (Liu et al., 2023; Guan et al., 2023; Xie et al., 2023). We refer to this line of work as "*Planner-Augmented LLMs*." Other similar approaches translate natural language into alternative representations of the planning problem, such as finite state automata (Yang et al., 2023a) or logic programming (Yang et al., 2023b), to solve them.

Benchmarking LLMs on Planning Tasks. Significant efforts have been made to develop benchmarks to assess the use of "*LLMs as Planners*." PlanBench (Valmeekam et al., 2023a) evaluates LLMs on various planning-related tasks, including plan generation, cost-optimal planning, plan verification, and others. Their work focuses on evaluating LLMs on their ability to generate plans. In contrast, *Planetarium* focuses on evaluating "*Planner-Augmented LLMs*". Here, the goal is to determine whether an LLM can successfully translate natural language descriptions of planning problems into the correct PDDL representations of those problems. This is arguably a more natural use of LLMs, but it is harder to evaluate because the output is an internal representation and not the end result of the planning process.

Planning problem equivalence. There is also work exploring techniques for determining the equivalence of planning problems, such as Shrinah et al. (2021). Chrpa et al. (2023) and Sievers et al. (2021) use a lifted graph representation of the planning domain, with the former specifically using graph isomorphism to check equivalence. Planetarium also employs graph isomorphism, but our focus is on evaluating the equivalence of PDDL problem instances rather than domains. Since domains and problem instances differ, the representation and isomorphism checks must also be adapted accordingly.

3 Preliminaries

To present *Planetarium*, we first introduce planning, PDDL, and scene graphs.

3.1 Classical Planning Problems

We use the set-theoretic form of classical planning problems (Ghallab et al., 2004).

Definition 1. A planning problem P is denoted by the tuple $(L, S, A, \gamma, s_i, g)$, where:

- L is a finite set of proposition symbols representing different facts about the world.

- $S \subseteq 2^L$ is a set of states. Each state $s \subseteq L$ is the set of true propositions in that state.

- A is the set of actions. Each action a is a triplet (pre(a), add(a), del(a), where pre(a) is the set of preconditions that must be satisfied for a to be executed, add(a) is the set of positive effects that become true after executing a, and del(a) is the set of negative effects that become false after executing a.

- The transition function $\gamma(s, a) = (s - \text{del}(a)) \cup \text{add}(a)$ models how the world changes by an action.

- s_i is the initial state of the world from which the problem begins.

- $g \subseteq L$ represents the goal propositions, indicating which propositions must be true for a state to be considered a goal state. The set of goal states is $S_g = \{s \in S \mid g \subseteq s\}$.

A plan $\pi = < a_1, a_n, \ldots, a_n >$ is a sequence of actions that leads from s_i to any state in S_g following γ. This plan is a *solution* to P.

3.2 Planning Domain Definition Language

PDDL is a specialized language designed to provide a unified way to represent planning problems. It can represent various types of planning problems, including classical planning problems (McDermott, 2000; Gerevini and Long, 2006). A PDDL planning problem consists of two files: the domain file, which describes the constant parts of the world model, and a problem file, which specifies particular planning scenarios using the world model outlined in the domain file.

The domain file specifies the set of possible actions A and their preconditions and effects, which

collectively define the transition function γ. The problem file defines the initial state s_i and the goal propositions g. Finally, the set of proposition symbols L is created by combining the predicates from the domain file with the objects defined in the problem file.

Planetarium focuses on classical planning problems within the STRIPS subset of PDDL (McDermott et al., 1998; Fikes and Nilsson, 1971). STRIPS provides the basic grammar for describing actions by specifying a set of preconditions that must be met for an action to be applicable and a set of effects that modify the propositions that are true after the action's execution. The expressiveness of problems defined in STRIPS is equivalent to those characterized by the set-theoretic definition (Definition 1) of classical planning problems (Ghallab et al., 2004).

3.3 Scene Graphs

To compare different planning problems, we use scene graphs. A scene graph is a data structure commonly used in fields such as computer vision and graphics (Johnson et al., 2015; Chang et al., 2023), rearrangement (Ramachandruni et al., 2023), and planning to represent objects, their attributes, and the relationships among them. In our work, we define scene graphs as directed graphs with types and attributes for both nodes and edges. We represent a PDDL problem file with scene graphs as follows. We create one scene graph (the *initial scene*) for the initial state and another for the set of goal propositions (the *goal scene*). For every object, we create a node with an `object` type. Then, for every proposition that is listed in the problem file, we create a node with a `proposition` type. That node is given an attribute with the name of its predicate. Then, for each argument to the predicate in that proposition, we add an edge from the proposition node to the corresponding object. Each edge is given three attributes: the name of the predicate, the position of the argument (first, second, etc.), and whether it is defined in the initial state or the goal propositions. A scene graph is thus $(\{O \cup P\}, E)$, where O is the set of nodes with `object` type, P is the set of nodes with `Proposition` type, and E is the set of edges.

We further define a *problem graph* as the combination of an initial scene and goal scene. Given $SceneGraph_{init} = (\{O \cup P_{init}\}, E_{init})$ and $SceneGraph_{goal} = (\{O \cup P_{goal}\}, E_{goal})$, then a problem graph merges the scene graphs such that

$ProblemGraph = (\{O \cup P_{init} \cup P_{goal}\}, \{E_{init} \cup E_{goal}\})$. We define graph isomorphism on any of these graphs as an edge and type-preserving bijection between nodes, meaning that two graphs share a connectivity structure where all types and attributes match. See Appendix B for examples of diagrams of scene and problem graphs.

4 Evaluation Framework

In this section, we describe the design of our benchmark. *Planetarium* consists of two components: an algorithm that validates whether two PDDL problem files, a ground truth file and an LLM-generated file, are equivalent, and a curated dataset of planning tasks against which an LLM can be evaluated.

4.1 Planning Problem Equivalence

The first step to benchmarking PDDL generation is determining how to decide whether the generated code matches ground truth code. One might assume that checking if two PDDL problem files are equivalent is straightforward. However, the same goal state could be represented by many PDDL problem files, as shown in Figure 1.

Given these difficulties, we propose a definition of equivalence in terms of classical planning problems. The main idea is to find a *bijective function* between the sets of proposition symbols L of the two problems such that makes the two problems equal. Our definition assumes that the transition function γ is shared between the two problems. Our formal definition of equivalence between two planning problems follows.

Definition 2. Two planning problems $P_1 = (L^1, S^1, A^1, \gamma, s_i^1, g^1)$ and $P_2 = (L^2, S^2, A^2, \gamma, s_i^2, g^2)$ with the same transition function γ are **equivalent** if there exists a bijective function $f : L^1 \to L^2$ such that:

1. $S^2 = \{\{f(p) : p \in s\} : s \in S^1\}$

2. $s_i^2 = \{f(p) : p \in s_i^1\}$

3. $S_{g^2} = \{\{f(p) : p \in s\} : s \in S_{g^1}\}$

4. $A^2 = \{(\quad \{f(p) : p \in \mathsf{pre}(a)\} \quad,$
$\{f(p) : p \in \mathsf{add}(a)\} \quad,$
$\{f(p) : p \in \mathsf{del}(a)\} \quad) : a \in A^1\}.$

That is, all predicates across the states, action definitions, initial state, goal specification, and actions of one problem are replaced with their corresponding predicates from the other.
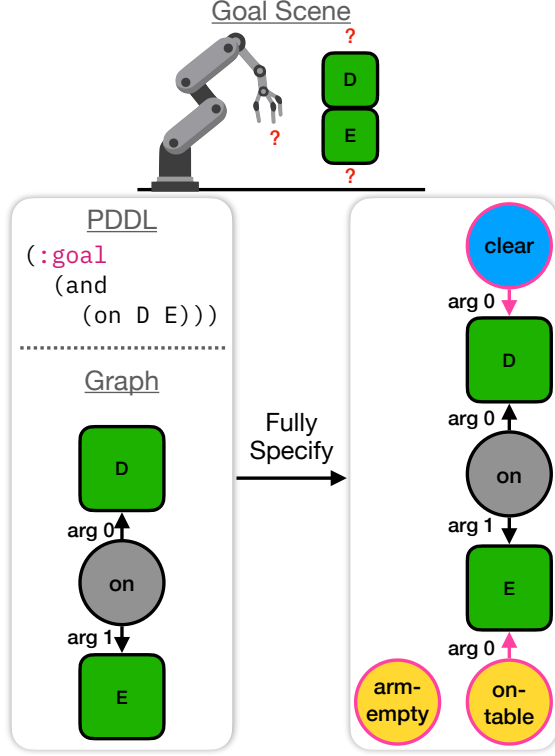
Figure 2: A demonstration of how `fullySpecify` fills in gaps of the goal state of a planning problem.

This definition is not directly usable for checking equivalence between two PDDL problem files because it relies on finding a bijection between the sets L of each problem, and PDDL does not define these sets directly. To build the set L with PDDL, one needs to use the predicates in the domain file and the objects in the problem file, instantiating each predicate with all possible objects they might take. We assume the entire PDDL domain is shared between the problems. This fact entails that the predicates will be shared, making it necessary to look for bijective functions for PDDL only over the objects. The challenge is that each PDDL problem file can correspond to many pairs of initial and goal states because the set of goal propositions makes an open-world assumption. That set can leave implicit trivial propositions that are necessarily true and therefore do not change the underlying planning problem. We must therefore *fully specify* the goal, meaning that we identify all propositions that are true in all reachable goal states when starting from the initial state.

Our algorithm for checking equivalence is summarized in Algorithm 1. First, we transform each set of initial state and goal state propositions into scene graphs. Second, we fully specify the goal scene graphs by adding all trivially true edges. Fi-

nally, we join the initial state scene graph with each goal state graph to create problem graphs and look for a bijection between objects such that the problem graphs are isomorphic.

Algorithm 1 Planning Problem Equivalence

1: **function** EQUIVALENT(P_a, P_b, isPlaceholder)
2:     $s_{i,a}, g_a \leftarrow$ toSceneGraph(P_a)
3:     $s_{i,a}, g_a \leftarrow$ toSceneGraph(P_b)
4:     **if** canDoFast($s_{i,a}, g_a, s_{i,b}, g_b$) **then**
5:         **return** fastEquivalent($s_{i,b}, g_a, s_{i,a}, g_b$)
6:     **end if**
7:     $g_a^\star \leftarrow$ fullySpecify($s_{i,a}, g_a$)
8:     $g_b^\star \leftarrow$ fullySpecify($s_{i,b}, g_b$)
9:     **if** isPlaceholder **then**
10:         init_iso $\leftarrow$ isIsomorphic($s_{i,a}, s_{i,b}$)
11:         goal_iso $\leftarrow$ isIsomorphic($g_a^\star, g_b^\star$)
12:         **return** init_iso $\wedge$ goal_iso
13:     **else**
14:         $p_a \leftarrow$ join($s_{i,a}, g_a^\star$)
15:         $p_b \leftarrow$ join($s_{i,b}, g_b^\star$)
16:         **return** isIsomorphic(p_a, p_b)
17:     **end if**
18: **end function**

Transform to Scene Graphs. From each PDDL problem file, we generate two scene graphs: one for the initial state and another for the goal state (lines 2–3). For each transformation, we first create an `object` node for each object. Then, for each proposition in the collection, we create a new `proposition` node and create edges between the proposition node and the objects it takes as arguments. Edge attributes denote argument order, predicate type, and whether the proposition is in an initial or goal scene. See Section 3.3 for details.

Check Easy Cases. For speed, we check several cases and return early if we can (lines 4–6). First, if the number of `object` nodes in each graph is *not* equal, the problems cannot be equivalent. Second, if the initial scenes are *not* isomorphic, then the problems cannot be equivalent. Finally, if the problem graphs, composed of the initial and goal scenes, are isomorphic, then the problems are equivalent.

Fully Specify the Goal Scenes. If the input is not an easy case, then we have to reason about the sets of goal states defined by the goal scenes (Figure 2). Condition 3 of Definition 2 requires that the sets of goal states that are consistent with the given goal propositions be equal after substi-

Table 1: A breakdown of the *Planetarium* dataset by the level of abstractness of the text description, number of propositions in the ground truth PDDL problem (size), and train/test split. Each instance consists of a ground truth PDDL problem description paired with a text description.

Domain	Total	Abstractness				Num. of Ground Truth Propositions (Size)					Split	
		Explicit to Explicit	Explicit to Abstract	Abstract to Explicit	Abstract to Abstract	1-20	21-40	41-60	61-80	>80	Train	Test
Blocks World	85,605	22,595	20,152	20,152	22,706	963	10,011	47,153	25,766	1,712	80,612	4,993
Gripper	54,141	13,683	10,565	12,466	17,427	2,334	13,640	16,405	20,325	1,437	49,363	4,778
Floor Tile	6,172	2,289	797	793	2,293	706	2,114	2,224	404	724	0	6,172

tuting matching propositions. Since PDDL uses the open-world assumption for goals, we have to identify all propositions that are true in all reachable goal states when starting from the initial state for each problem. The function `fullySpecify` finds all such propositions and adds them to the goal scenes as additional `proposition` nodes with the corresponding edges and attributes (lines 7–8). We show how to implement `fullySpecify` efficiently for the three domains in *Planetarium* in Appendix C.

PDDL without Object Identity. Our algorithm runs in two modes, depending on the type of problem it is checking. Sometimes we want to compare a generated PDDL problem file with many ground truth problem files and see if the generated file matches any one of them. For example, if the natural language description says "make a tower of height 3," the specific blocks to use are unspecified, and any permutation of blocks that builds a tower of height 3 should be considered correct. Concretely, we want to treat the objects in the PDDL goal states as *placeholders* and accept any permutation of them. We check this condition when `isPlaceholder` is **True** (determined by the problem type) simply by checking isomorphism between initial and goal scenes separately rather than combining them into problem graphs (lines 10–12).

PDDL with Object Identity. If isPlaceholder is **False**, we want to check whether the problem files are precisely equivalent under Definition 2, meaning that the objects in the initial scenes correspond to the same objects in the goal scenes as well. We first join the corresponding initial and scene graphs into problem graphs as described in Section 3.3 (lines 14–15). Then we check whether the two problem graphs are isomorphic (line 16).

We illustrate Algorithm 1 with examples in Appendix B. We also formally state its correctness, with the proof in Appendix A.

Theorem 1. Equivalent(P_a, P_b, **False**) *returns* **True** *if and only if the PDDL problem files P_a and P_b represent equivalent planning problems under Definition 2.* Equivalent(P_a, P_b, **True**) *returns* **True** *if and only if P_a represents a planning problem that is equivalent to some planning problem represented by P_b after a permutation of the objects in its goal state.*

4.2 The Dataset

The *Planetarium* dataset includes 145,918 text-to-PDDL pairs derived from the Blocks World, Gripper, and Floor Tile domains (Seipp et al., 2022; McDermott, 2000). Our dataset captures 25 unique initial and goal state configurations describing 73 different tasks. In the Blocks World domain, a robotic hand manipulates blocks arranged in various configurations on a table, with 5 predicates and 4 actions. The Gripper domain features a robot with grippers transporting balls between rooms, with 7 predicates and 3 actions. While versions of Blocks World and Gripper domains exist with `:typing`, we choose to use the IPC standard versions, which do not. We simplified the standard Floor Tile domain (Seipp et al., 2022), where robots are tasked with painting tiles, by removing the `left` and `down` predicates and relying on the `right` and `up` predicates to check relative positions. We further relax the domain by removing the `clear` predicate (which introduces path constraints) and the `free-color` predicate (which IPC problems did not use and is not modified or required by any actions in the original domain). Finally, we added `paint-left` and `paint-right` actions, which reduced edge cases for the domain, resulting in a domain with 6 predicates and 9 actions (compared to 10 predicates and 7 actions in the IPC version). More details about the dataset can be found in Appendix D.

Table 2: Examples of abstract and concrete descriptions for each domain in the benchmark.

Domain	Abstract	Explicit
blocksworld	You have 3 blocks, stacked into 2 towers of heights 1, 2, and your arm is empty. Your goal is to invert each individual stack of blocks, such that the block that in each tower that was originally on the bottom will be on the top.	Your arm is empty. b1 is clear. b1 is on the table. b2 is clear. b2 is on b3. b3 is on the table. Your goal is to have the following: Your arm should be empty. b3 should be clear. b3 should be on b2. b2 should be on the table. b1 should be clear. b1 should be on the table.
gripper	You have 2 rooms, 2 balls, and 2 grippers. 1 balls are distributed across the same number of grippers, and the rest are in the first room. The robby is in the first room. Your goal is to gather all balls into one room.	You have 2 rooms, 2 balls, and 2 grippers. 1 balls are distributed across the same number of grippers, and the rest are in the first room. The robby is in the first room. Your goal is to have the following: gripper1 should be free. gripper2 should be free. ball1 should be at room1. ball2 should be at room1.
floor-tile	You have 1 robots, 2 colors, and 2 unpainted tiles arranged in a grid with 1 rows and 2 columns. The first robot is at the top-left corner, and has the first color. All colors are available. Your goal is to paint all the tiles with the same color.	You have 1 robot. You have 2 tiles. You have 2 colors. Tile tile2 is to the right of tile tile1. The robot robot1 has color color1. The robot robot1 is at tile tile1. Color color1 is available. Color color2 is available. Your goal is to have the following: Tile tile1 should be painted with color color1. Tile tile2 should be painted with color color1.

Dataset Construction. We focus on these three domains commonly used in other works (Valmeekam et al., 2023a; Liu et al., 2023) but are nevertheless challenging. Each entry in our dataset is a text-to-PDDL pair consisting of a text description detailing the initial and goal states and the corresponding ground truth PDDL. (See Appendix H for examples.) The composition of the *Planetarium* dataset is presented in Table 1, which categorizes the data according to text description abstractness and the number of propositions. The table further delineates the train/test split, illustrating the distribution of data reserved for evaluation purposes. For a comprehensive overview of the task descriptions and the specific allocation of tasks between training and test sets, readers are directed to Appendix D. The dataset is procedurally generated: we handcraft templates for each task configuration, which we combine with one another to generate problems at scale.

We vary the data along two dimensions: abstractness (explicit vs. abstract) and size. Explicit planning problem text descriptions correspond directly to propositions found in the problem PDDL (e.g., "block 1 is on block 2"). Abstract text descriptions instead summarize a state (e.g., "all blocks are in a single tower").

Since our text descriptions contain both initial and goal states, each can be either an abstract or explicit description. This leads to four possible abstractness categories: explicit to explicit, explicit to abstract, abstract to explicit, and abstract to abstract.

We measure the size of a problem by the number of propositions listed in the ground truth problem PDDL. Larger problems typically pose greater challenges for LLMs.

5 Evaluating LLMs on *Planetarium*

As an initial snapshot of the field's current state, we evaluate several API-access and open-weight language models on *Planetarium* in both zero-shot and fine-tuned settings. We find that while powerful models like GPT-4o can often generate valid PDDL problems (in the sense that some plan solves them), they are rarely correct. This result underscores the need for *Planetarium*'s rigorous approach to evaluation of PDDL generation. The code to recreate the entire evaluation is available at github.com/BatsResearch/planetarium.

Models. We evaluate two API-access models, GPT-4o and o1-mini, and we evaluate three open-weight models before and after fine-tuning: Gemma 2 2B IT, Gemma 2 9B IT, and Gemma 2 27B IT (Team, 2024). Details regarding fine-tuning can be found in Appendix F.

Evaluation Protocol. We evaluate models on the *Planetarium* test set, which consists of heldout configurations from the Blocks World and Gripper domains and the entirety of the Floor Tile domain. Models are prompted with the natural language description of the task along with the respective domain PDDL. We record three metrics for the generated problems: the number of parseable problems, the number of solvable problems, and the number
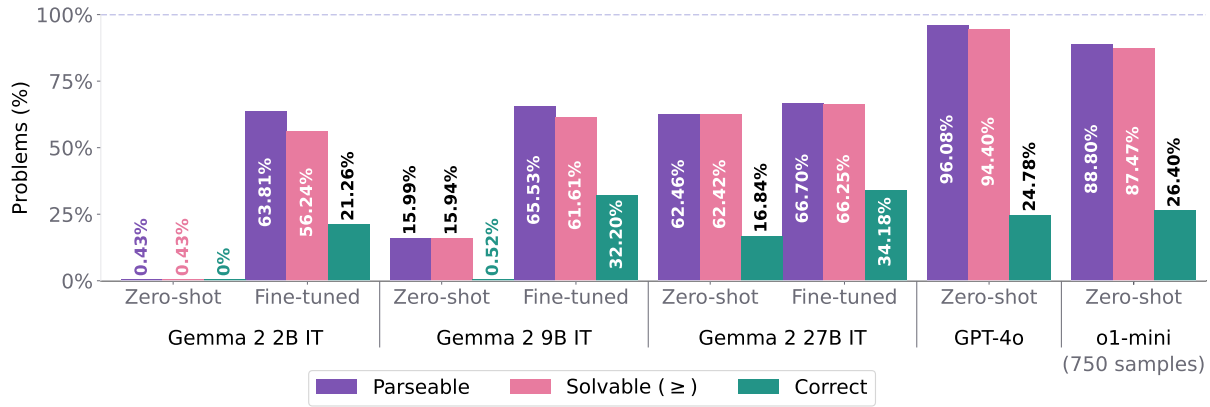
Figure 3: Performance of various models on the *Planetarium* test set.

of correct problems. We say a model output is *parseable* if a PDDL parser supporting `:strips` can extract a valid PDDL problem from a substring in the output and if it can be converted into our graph representation. A problem is *solvable* if it is parseable and a plan can be applied to the initial scene that results in the goal scene. Due to the size and complexity of some of the problems in our dataset, a generalized classical planner cannot always reliably and quickly return solutions. Instead, we built specialized planners for Blocks World and Gripper problems that work on all problems in our dataset and generally all validly defined `blocksworld` and `gripper` domain problems except for a few invalid edge cases (e.g., one block on top of two blocks at a time, holding two blocks, etc.). We use Fast Downward (Helmert, 2006) for `floor-tile` problems. We then validate all plans with VAL (Howey et al., 2004). We then feed parseable and solvable problems into our PDDL equivalence algorithm to verify equivalence to our ground truth PDDL and thereby determine if it is *correct*. We found our equivalence algorithm to take, on average, 12ms per example to compute on an M2 Apple Silicon laptop with batch parallelization.

Results. The performance of Gemma 2 2/9/27B IT, GPT-4o, and o1-mini on the *Planetarium* test set are shown in Figure 3. We report the percentage of generated plans that were parseable, solvable, and correct from zero-shot and fine-tuned settings, averaged across all domains. For o1-mini, we randomly sample 250 problems per domain to evaluate due to prohibitive costs. In the zero-shot setting, o1-mini and GPT-4o performed the best, followed closely by Gemma 27B IT. For all models, the per-
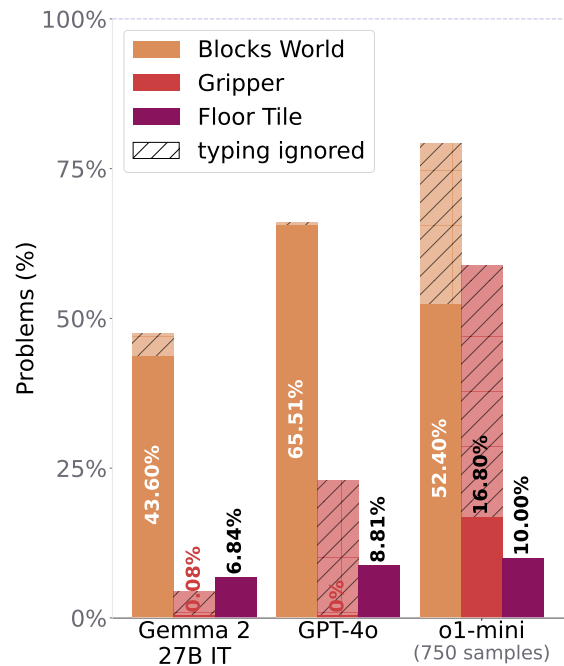

Figure 4: Breakdown of zero-shot performance by domain for Gemma 2 27B IT, GPT-4o, and o1-mini.

centage of generated PDDL that is semantically correct significantly lags behind the percentage of generated PDDL that is parseable and solvable.

Fine-tuning improved performance across all open-weight models, with fine-tuned Gemma 2 27B IT achieving the highest accuracy of all tested models at 34.2%.

Takeaways We find two key takeaways from the type of mistakes these language models tended to make. First, a significant portion of errors come from models "ignoring" the domain in context and referring to an incorrect domain, likely from its prior training corpus. We can see this by the large increase in accuracy when ignoring `:typing`,

shown in Figure 4: our Blocks World and Gripper domains do not have typing, yet when generating PDDL, these models confidently and consistently add object types.

Second, there seems to be a strong correlation between a model's performance on a certain domain and the domain's frequency in the wild: Gripper and Blocks World are often used as introductory PDDL examples, while Floor Tile is less common. However, Gripper, and to a lesser extent, Blocks World, is often defined with `:typing`. This may explain the difference in performance between Blocks World and Gripper across models, as well as the difference in performance between enforcing versus not enforcing `:typing`.

Succinctly, we see a trend when relaxing the `:typing` requirement that models find problems in less common domains harder, accounting for everything else.

6 Conclusion

Planetarium is a new benchmark for assessing the ability of LLMs to translate natural language descriptions of planning problems into PDDL. Our work reveals that language models find it challenging to generate semantically correct structured planning language descriptions. Models like GPT-4o can often produce valid, seemingly correct descriptions (94.4%) when, in reality, only a small fraction (24.8%) are genuinely correct. We hope that *Planetarium* will drive progress on hybrid approaches combining LLMs and classic planners, setting a standard for evaluating such tasks.

Acknowledgements

Limitations

Planetarium has a few important limitations: *Planetarium* currently only supports the Blocks World, Gripper, and Floor Tile domains. While these domains have been popular for studying LLMs and their relation to planning, incorporating more expressive domains in the future will widen the scope

of *Planetarium*. This benchmark is also currently restricted to the STRIPS subset of PDDL. Extending it to support more expressive subsets of PDDL will allow us to evaluate more complex, real-world planning problems such as non-deterministic, temporal, and numeric domains.

Potential Impacts

A potential societal impact of this research is ensuring the correctness of translating natural language into structured planning languages. If this translation method becomes widespread but its evaluation remains inaccurate, systems could produce misleading or misaligned results that could cause harm if acted upon. *Planetarium* highlights the importance of assessing the correctness of translations. We achieve this by reasoning about PDDL semantics and the inherent structure of classical planning problems.

References

Jorge A. Baier, Fahiem Bacchus, and Sheila A. McIlraith. 2009. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence*, 173(5):593–618. Advances in Automated Plan Generation.

Georgia Chalvatzaki, Ali Younes, Daljeet Nandha, An Thai Le, Leonardo FR Ribeiro, and Iryna Gurevych. 2023. Learning to reason over scene graphs: a case study of finetuning GPT-2 into a robot language model for grounded task planning. *Frontiers in Robotics and AI*, 10.

Xiaojun Chang, Pengzhen Ren, Pengfei Xu, Zhihui Li, Xiaojiang Chen, and Alex Hauptmann. 2023. A comprehensive survey of scene graphs: Generation and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):1–26.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya

Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Lukáš Chrpa, Carmine Dodaro, Marco Maratea, Marco Mochi, and Mauro Vallati. 2023. Comparing planning domain models using answer set programming. In *Logics in Artificial Intelligence: 18th European Conference, JELIA 2023, Dresden, Germany, September 20–22, 2023, Proceedings*, page 227–242, Berlin, Heidelberg. Springer-Verlag.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized LLMs. *Preprint*, arXiv:2305.14314.

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

A. Gerevini and D. Long. 2006. Plan constraints and preferences in PDDL3. In *ICAPS Workshop on Soft Constraints and Preferences in Planning*.

Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated planning*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann, Oxford, England.

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. *arXiv preprint arXiv:2305.14909*.

Malte Helmert. 2006. The fast downward planning system. *J. Artif. Int. Res.*, 26(1):191–246.

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.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022a. Language models as zeroshot planners: Extracting actionable knowledge for embodied agents. ArXiv:2201.07207.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and brian ichter. 2022b. Inner monologue: Embodied reasoning through planning with language models. In *6th Annual Conference on Robot Learning*.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown,

Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. 2022. Do as I can, not as I say: Grounding language in robotic affordances. In *CoRL*, pages 287–318.

Justin Johnson, Ranjay Krishna, Michael Stark, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. 2015. Image retrieval using scene graphs. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3668–3678.

Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. 2019. SPoC: Search-based pseudocode to code. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Carlos Linares López, Sergio Jiménez Celorrio, and Ángel García Olaya. 2015. The deterministic part of the Seventh International Planning Competition. *Artificial Intelligence*, 223:82–119.

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

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. 1998. PDDL – The planning domain definition language. *Technical Report, Tech. Rep.*

Drew M. McDermott. 2000. The 1998 AI Planning Systems Competition. *AI Magazine*, 21(2):35.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, ACL '02, page 311–318, USA. Association for Computational Linguistics.

Kartik Ramachandruni, Max Zuo, and Sonia Chernova. 2023. ConSOR: A context-aware semantic object rearrangement framework for partially arranged scenes. *Preprint*, arXiv:2310.00371.

Allen Z. Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, Zhenjia Xu, Dorsa Sadigh, Andy Zeng, and Anirudha Majumdar. 2023. Robots that ask for help: Uncertainty alignment for large language model planners. In *7th Annual Conference on Robot Learning*.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. CodeBLEU: a method for automatic evaluation of code synthesis. *Preprint*, arXiv:2009.10297.

Baptiste Roziere, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised translation of programming languages. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

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

Pratyusha Sharma, Antonio Torralba, and Jacob Andreas. 2022. Skill induction and planning with latent language. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1713–1726, Dublin, Ireland. Association for Computational Linguistics.

Anas Shrinah, Derek Long, and Kerstin Eder. 2021. D-VAL: An automatic functional equivalence validation tool for planning domain models. *CoRR*, abs/2104.14602.

Silvan Sievers, Gabriele Röger, Martin Wehrle, and Michael Katz. 2021. Theoretical foundations for structural symmetries of lifted PDDL tasks. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29(1):446–454.

Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B. Tenenbaum, Leslie Pack Kaelbling, and Michael Katz. 2023. Generalized planning in PDDL domains with pretrained large language models. In *PRL Workshop Series – Bridging the Gap Between AI Planning and Reinforcement Learning*.

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

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2022. ProgPrompt: Generating situated robot task plans using large language models. *CoRR*, abs/2209.11302.

Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fišer, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp. 2024. The 2023 International Planning Competition. *AI Magazine*.

Gemma Team. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Mauro Vallati, Lukas Chrpa, Marek Grześ, Thomas Leo McCluskey, Mark Roberts, Scott Sanner, and Managing Editor. 2015. The 2014 International Planning Competition: Progress and trends. *AI Magazine*, 36(3):90–98.

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

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023b. On the planning abilities of large language models - a critical investigation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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

Vincent Vidal and Héctor Geffner. 2006. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artificial Intelligence*, 170(3):298–335.

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*.

Yunhao Yang, Jean-Raphael Gaglione, Cyrus Neary, and ufuk topcu. 2023a. Large language models for verifiable sequential decision-making in autonomous systems. In *2nd Workshop on Language and Robot Learning: Language as Grounding*.

Zhun Yang, Adam Ishay, and Joohyung Lee. 2023b. Coupling large language models with logic programming for robust and general reasoning from text. *arXiv preprint arXiv:2307.07696*.

A   Proof of Theorem 1

Proposition 1. Equivalent(P_a, P_b, *False*) *returns* *True* *if and only if the PDDL problem files* P_a *and* P_b *represent equivalent planning problems under Definition* 2.

Proof. To prove this proposition, we need to prove each direction of the biconditional. First, we will prove the forward implication. There are two possible ways the algorithm can return `True`: Either in the `isIsomorphic` check (line 16) or in `fastEquivalent` (line 5).

Case 1: When `isIsomorphic(pₐ, p_b)` returns `True`, we know that there exists a bijection φ from vertices V_{p_a} to vertices V_{p_b}. Since φ is a type-preserving bijection, it means the two graphs share a connectivity structure where all types and attributes match.

By using Lemma 1, we can build a new bijection f from φ. Instead of mapping vertices in the scene graph, f maps propositions from L^a in planning problem P_a to propositions in L^b in planning problem P_b. Given that φ is a type-preserving bijection, we know that the bijection f derived from φ also maps objects to objects, predicates to predicates, vertices in the initial scene $s_{i,a}$ to vertices in $s_{i,b}$, and vertices in the goal scene g_a^* to vertices in g_b^*.

To prove Property 1, we can build the set L^a for each planning problem by taking each predicate in the PDDL domain and instantiating it with all possible combinations of objects from the problem PDDL P_a. Then, we can construct the set of all possible states of the problem by taking the power set $S^a = 2^{L^a}$. For each proposition $p \in s$ in each state $s \in S^a$, we apply the bijection f to show that we get S^b:

$$
\begin{aligned}
S &= \{\{f(p) : p \in s\} : s \in S^a\} \\
&= \{\{\langle \varphi(q^a), \varphi(o_1^a), \ldots, \varphi(o_k^a) \rangle : p \in s\} : s \in S^a\} \\
&= \{\{\langle q^b, o_1^b, \ldots, o_k^b \rangle : p \in s\} : s \in S^a\} \\
&= \{\{p_b : p \in s\} : s \in S^a\} \\
&= S^b.
\end{aligned}
$$

To prove Property 2, we go through each proposition p in the initial state s_i^a of the problem PDDL P_a, and apply the bijection f to show that we get s_i^b:

$$
\begin{aligned}
s_i &= \{f(p_a) : p_a \in s_i^a\} \\
&= \{\langle \varphi(q^a), \varphi(o_1^a), \ldots, \varphi(o_k^a) \rangle : p_a \in s_i^a\} \\
&= \{\langle q^b, o_1^b, \ldots, o_k^b \rangle : p_a \in s_i^a\} \\
&= \{p_b : p_a \in s_i^a\} \\
&= s_i^b.
\end{aligned}
$$

To prove Property 3, we first observe that the `fullySpecify` function adds all possible trivial propositions to the goal scene graph. By definition, trivial propositions are those that are known not to change the underlying planning problem when added or removed. We can use the same bijection f constructed from φ. We don't need to add or remove non-trivial predicates, as doing so would change the planning problem by definition. This implies that the augmented goal state represents any other possible goal state in S_g. With this, we can use the bijection f built from φ to apply it to any state in S_{g^a} and show that we get S_{g^b}:

$$
\begin{aligned}
S_g &= \{\{f(p) : p \in s\} : s \in S_{g^a}\} \\
&= \{\{\langle \varphi(q^a), \varphi(o_1^a), \ldots, \varphi(o_k^a) \rangle : p \in s\} : s \in S_{g^a}\} \\
&= \{\{\langle q^b, o_1^b, \ldots o_k^b \rangle : p \in s\} : s \in S_{g^a}\} \\
&= \{\{p_b : p \in s\} : s \in S_{g^a}\} \\
&= S_{g^b}.
\end{aligned}
$$

Finally, to prove Property 4, we observe that Equivalent(P_a, P_b, **False**) assumes that the domain files are shared between P_a and P_b. As a result, any bijection φ found by the algorithm must preserve the equality of actions across both problems, since the shared domain enforces consistency in their action sets.

Case 2: We can use the same arguments from Case 1 with one slight modification. Here, `fastEquivalent` returns `True` only if `isIsomorphic(join($s_{i,a}$, $g_a$)`, `join($s_{i,b}$, $g_b$))` returns `True`. The difference with Case 1 is that we join the initial states with the goal propositions scene graph g instead of the fully specified scene graph $g^\star$.

In Case 1 we have proven that when `isIsomorphic(pₐ, p_b)` returns `True`, the planning problems P_a and P_b are equivalent under the bijection f. Since $p_a = $ `join($s_{i,a}$, $g_a^\star$)` and $p_b = $ `join($s_{i,b}$, $g_b^\star$)`, we can let $g_a^\star = g_a$ and $g_b^\star = g_b$. This is because, by definition, trivial propositions do not change the underlying planning problem

when added or removed, and in this case, we are only removing trivial propositions. Hence, by the proof of Case 1 we conclude that this case has also to be true.

Now, we will prove the backward implication. In this case, we will assume that there exists a bijection f that follows properties 1, 2, and 3 of Definition 2. By Lemma 1, we know that f comes from φ, and since both are bijections, we can recover φ by doing the inverse process. We need to prove that these conditions lead to isIsomorphic (line 16) or fastEquivalent (line 5) returning True.

Since there exists a bijection φ that comes from f, the number of objects must be the same for planning problems P_a and P_b, as stated in Property 1. With Property 2, we know that this bijection φ must also make the initial scenes isomorphic. Assume that the sets of goal propositions have the same size. By using Property 3, we know that the bijection φ would make fastEquivalent (line 5) return True.

Conversely, assume that the sets of goal propositions have different sizes. In this case, canDoFast would not allow fastEquivalent to execute. In the case of isIsomorphic (line 16), we know that $p_a = \text{join}(s_{i,a}g_a^\star)$ and $p_b = \text{join}(s_{i,b}, g_b^\star)$. Since we know that we can construct the bijection φ that maps vertices from the bijection f, we also know that since φ comes from f, this bijection respects initial states (Property 2) and goal states (Property 3). Furthermore, the algorithm does not modify any actions in the domain file, and since a single domain is shared across all problems, the actions remain consistent between them (Property 4). Therefore, isIsomorphic(p_a, p_b) must return True.

With this, we have proven that if two problems are equivalent under Definition 2, then the equivalence algorithm must return True.

Since we have proven both directions of the biconditional, Proposition 1 is true. □

Proposition 2. Equivalent(P_a, P_b, ***True***) *returns **True** if and only if P_a represents a planning problem that is equivalent (under Definition 2) to some planning problem represented by P_b after a permutation of the objects in its goal state.*

Proof. To prove this proposition, we need to prove each direction of the biconditional. First, we will prove the forward implication. There are two possible ways the algorithm can return True: Either in fastEquivalent (line 5) or in isIsomorphic($s_{i,a}$, $s_{i,b}$) $\land$ isIsomorphic($g_a^\star$, $g_b^\star$) (line 12).

To prove the case where fastEquivalent (line 5) returns True, we can use the same argument as in Case 1 of the forward implication in the proof of Proposition 1. This is because the placeholder being True does not affect the fastEquivalent function.

Now, assume that isIsomorphic($s_{i,a}$, $s_{i,b}$) $\land$ isIsomorphic($g_a^\star$, $g_b^\star$) is True. This implies that there exist two bijections: one φ on the vertices $V_{s_{i,a}}$ to $V_{s_{i,b}}$, and another ψ between vertices $V_{g_a^\star}$ to $V_{g_b^\star}$. Now, by using Lemma 1, we can build two new bijections, f and h, that map between propositions in L instead of nodes in the scene graph, as φ and ψ do.

Since the underlying mappings of f and h can be different, we need a way to combine them into a single bijection. One way to construct this bijection is to permute the mappings of f until they fit the mappings of h. This implies that there are many pairs of f and h, and each pair can be combined into a global bijection using function composition $f \circ h$. For each of these permutations, we can then use the arguments in Case 2 of the proof of Proposition 1 to show that these pairs hold all three properties of Definition 2.

Now, we will prove the backward implication. First, since fastEquivalent remains the same, we do not need to prove anything new about it; we can simply reuse the previous results.

Assume P_a represents a planning problem that is equivalent to some planning problem represented by P_b after a permutation of the objects in its goal state. This implies that there are several bijections f from the predicates in P_a to the predicates in P_b. For each of these bijections f, we can use Lemma 1 to transform it into φ. Since we know by Property2 that φ maps initial states, there is an isomorphism between the initial states. By Property 3, we know that φ maps any goal state of P_a to any other possible state in P_b. Hence, $s_a^\star$ has to be isomorphic with $s_b^\star$ since φ is a bijection between their nodes.

Since we have proven both directions of Proposition 2, we know it is true. □

Proof of Theorem 1. Having the proof of Proposition 1 and the proof of Proposition 2, we know that the theorem, which is just the conjunction of these propositions, must be true. □

Lemma 1. *Let φ be an isomorphic bijection between nodes in two scene graphs (propositions or objects). There exists a bijection f over the propositions in L such that for any proposition $p \in L$,*

$f(p) = \langle \varphi(q), \varphi(o_1), \varphi(o_2), \ldots, \varphi(o_k) \rangle$, *where q is the predicate and $O = \{o_1, o_2, \ldots, o_k\}$ is a set of objects.*

Proof. Take any proposition $p \in L$ and identify its predicate q and the ordered set of objects it acts on, $O = \langle o_1, o_2, \ldots, o_k \rangle$. Then, we can transform the proposition p into the ordered tuple $p = \langle q, o_1, o_2, \ldots, o_k \rangle$. Finally, we can build $f(p) = \langle \varphi(q), \varphi(o_1), \varphi(o_2), \ldots, \varphi(o_k) \rangle$, which is a bijection over the propositions in L. $\qquad\square$

B Examples of Algorithm 1

Figure 5 shows examples of how Algorithm 1 checks whether two PDDL problems are equivalent.

C Implementation of `fullySpecify`

The `fullySpecify` function adds trivial edges (edges that must exist but are not currently present) to our scene graphs. We build the `fullySpecify` function using our domain knowledge of the Blocks World and Gripper domains.

Blocks World The following helpful facts are true about Blocks World:

- If all blocks have its behavior above it defined (they are either **clear**, have something **on** top of it, or is being held), then any block that does not have its bottom behavior specified, must be on the table (**on-table**).

- If all blocks have its behavior beneath it defined (they are either **on** top of something else, are on the table (**on-table**), or are being held), then any block that does not have its behavior above it defined must be **clear**.

- Following the last two facts: if there is a block **A** which has its behavior above it unspecified and block **B** which has its behavior below it unspecified, then the only way that (**on B A**) can exist is if there isn't already a chain of predicates that leads **A** to **B** or vice versa. If there already exists a chain connecting these two blocks, that means **A** and **B** are already connected by being in the same tower, and that they are the top and bottom of the tower they are a part of.

- If there are no "floating" blocks (blocks that have both its top and bottom behavior defined), and the arm's behavior is undefined (neither

arm-empty nor **holding**) is present, then the arm must be empty. This is because there is no block that the arm could possibly hold here.

Using these rules, we can add all possible trivial edges, until no more trivial edges can be found. Further, these facts will discover all possible trivial edges in Blocks World, meaning the fully specified scene graph will be in its canonical form after `fullySpecify`.

Gripper Similarly, we can build a set of facts that operate on the Gripper domain:

- If all balls are assigned a room (**at ball room**), then all unassigned grippers (no (**free gripper**) or (**carry ball gripper**)) must be **free**.

- If there is only one room, and if all grippers are already specified (either (**free**) or (**carry ball gripper**)), then any unspecified balls must be in the only room that exists.

- If there is only one room, the robby must be in it (**at-robby onlyRoom**).

These are the only rules we can find. This is because, if a ball's position is unspecified, it can always be assigned to any arbitrary room. If the robby is unspecified (no **at-robby**), then it too, can always be specified to any arbitrary room.

Floor Tile In the standard Floor Tile, we know that:

- A tile cannot be unpainted.

- All available colors (**available-color color**) are unchangeable and cannnot be added.

- Relative positional propositions (**right tile1 tile2**) and (**up tile1 tile2**) are unchangeable and cannot be added.

Further, given our simplifications, we have the following characteristics in our domain:

- Traversal is bidirectional.

- Painting a tile and other robots do not block path-finding.

- From the last two facts, robots cannot block themselves off through actions, so the only way the position of a robot is fixed is if:
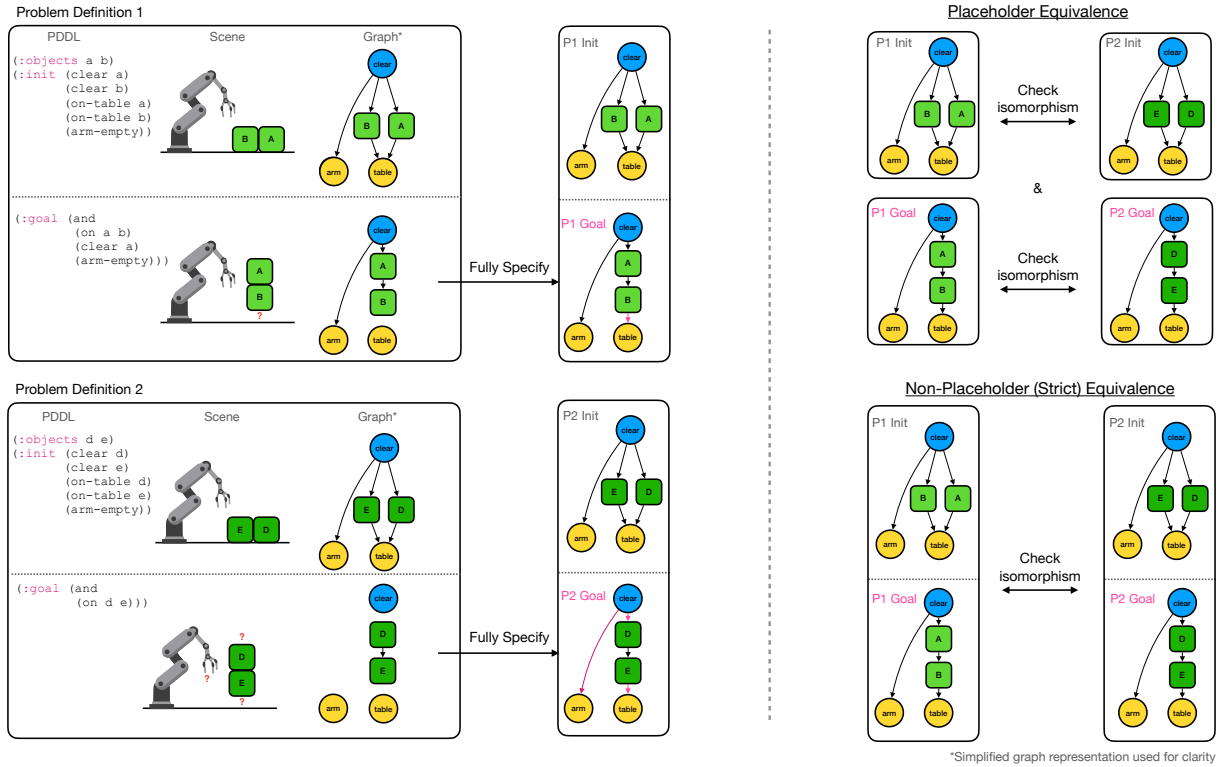
Figure 5: An illustration of the algorithm to check if two PDDL problems are equivalent. It shows each of the stages of the algorithm: transforming to scene graphs, fully specifying the goal propositions, and checking for graph isomorphism.

1. It is explicitly defined in the goal description.

2. The robot is on an isolated tile to begin with.

Applying these rules allows us to fully specify a Floor Tile goal description.

D   Dataset Details

To obtain interesting problems for each domain, we crafted a set of tasks instead of randomly generating instances. Below are descriptions of each domain and their corresponding task configurations.

Blocks World.   This domain involves using a robotic hand to manipulate a set of blocks arranged in various configurations on a table. Blocks World configurations:

- **Stacked**: All the blocks are in a single tower.

- **Unstacked**: All blocks are separately placed on the table.

- **Holding a Block**: All blocks except one are separately placed on the table, the robot holds the last block.
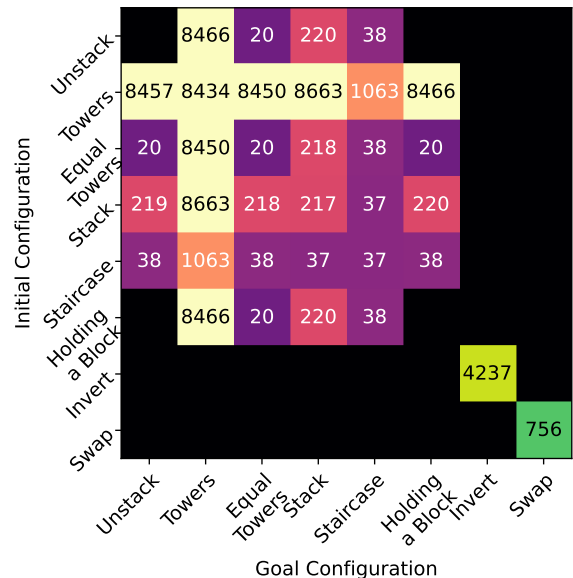

Figure 6: A visualization of the different tasks in Blocks World. Training set displayed in "magma" color map (purple to yellow). Test set displayed in "viridis" color map (blue to yellow).
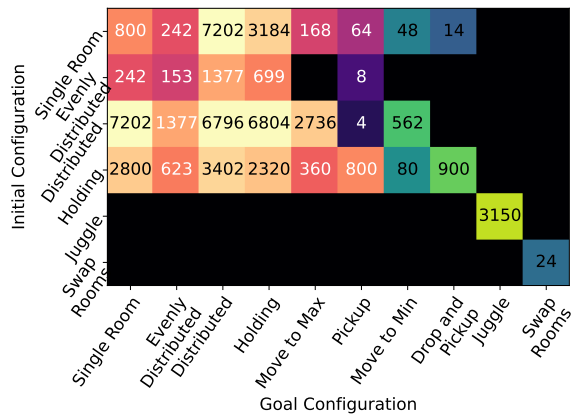
Figure 7: A visualization of the different tasks in Gripper. Training set displayed in "magma" color map (purple to yellow). Test set displayed in "viridis" color map (blue to yellow).

- **Staircase**: Towers are in incrementing heights (1, 2, 3, . . .).

- **Equal Towers**: Blocks are in towers of equal height.

- **Swap** (tied): Goal and initial configurations are tied: given two towers, swap the base blocks, and leave the rest of the structure unchanged.

- **Invert** (tied): Goal and initial configurations are tied: Given a set of stacks, flip each stack.

- **Towers**: Build towers to match a specified list of heights.

Gripper. This domain features a robot equipped with grippers, tasked with transporting balls from one room to another. Gripper Configurations:

- **Single Room**: All balls are in one room.

- **Evenly Distributed**: Each room has the same number of balls.

- **Distribute**: Each room has a predefined number of balls.

- **Swap Rooms** (tied): Goal and initial configurations are tied: swap ball locations between two rooms.

- **Move to Max** (goal only): Move all balls to the room that started with the most number of balls.

- **Move to Min** (goal only): Move all balls to the room that started with the fewest balls.
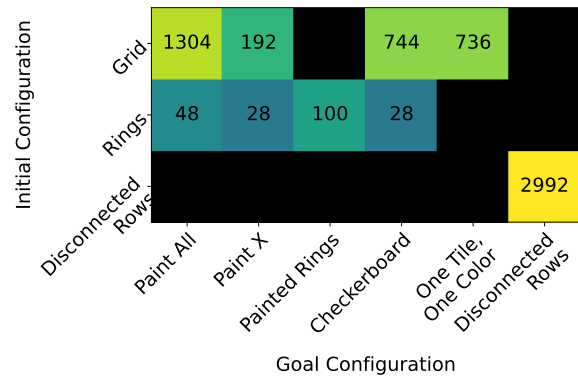

Figure 8: A visualization of the different tasks in Floor Tile. Training set displayed in "magma" color map (purple to yellow). Test set displayed in "viridis" color map (blue to yellow).

- **Pickup** (goal only): Pick up all the balls

- **Drop and Pickup** (goal only): Drop all the balls in the first room, and pick up all the balls that started in rooms.

- **Juggle** (tied): Goal and initial configurations are tied: given that the robot is holding some balls, shuffle which arm holds which ball.

Floor Tile. This domain features a group of robots tasked with painting some tiles. Floor Tile configurations:

- **Grid** (initial only): Tiles are arranged in a grid.

- **Rings** (initial only): Tiles are arranged in a grid in concentric rings.

- **Disconnected Rows** (tied): Goal and initial configurations are tied: tiles are in rows, but the rows are disconnected from one another. The goal is to paint both ends of each row.

- **Painted Rings** (goal only): Tiles are arranged in a grid in concentric rings, and each ring is painted.

- **Paint All** (goal only): All tiles are painted with one color.

- **Painted X** (goal only): The robot must paint an X across all the tiles.

- **One Tile, One Color** (goal only): Each tile is painted a different color.

- **Checkerboard** (goal only): The tiles are painted in a checkerboard pattern with two colors.

11238

E   License Information

The data for *Planetarium* is available on Hugging-Face[1] and released under a Creative Commons CC-BY-4.0 license. In addition, all code to create the dataset and evaluate models on the benchmark is available on GitHub[2] and released under a BSD 3 license. The authors are responsible for the content of the dataset.

In our study, we use the Blocksworld, Gripper, and Floor Tile domains, which are sourced from the IPC (McDermott, 2000). The specific domain files used in our research are obtained from `pddl-generators` (Seipp et al., 2022), which is distributed under the GNU General Public License. The domain file for Floor Tile is a modified version of the one found from `pddl-generators` (Seipp et al., 2022)

GitHub Copilot was used to assist in writing the code for this paper.

F   Fine-tuning

Hyperparameters	Value
Optimizer	adamw_torch
Learning rate	2e-5
Batch Size	1
Betas	(0.9, 0.999)
Epsilon	1e-8
Weight Decay	0.01
Max Sequence Length	1500
LoRA rank	16
LoRA alpha	32
LoRA Dropout	0.05

Table 3: Training hyperparameters when fine-tuning all models included in the paper. The hyperparameters are separated into three parts: supervised fine-tuning, generation-related, and LoRA-related.

We fine-tuned the open-weight models using QLoRA with a rank of 16, adhering to the hyperparameter recommendations for small models provided by the original authors (Dettmers et al., 2023). Models were loaded with 4-bit precision and trained over the training set for a single epoch. Fine-tuning for each model used either two NVIDIA GeForce RTX 3090 GPUs or two NVIDIA A6000 GPUs, operating with data parallelization for approximately 15 hours. We truncate the longest 5% of our training dataset due to GPU memory constraints.

Table 3 displays the hyperparameters used for fine-tuning across all models. In addition, models are loaded using 4-bit NF4 quantization with double quantization to reduce the average memory footprint. We also use the bfloat16 compute data type for faster training.

G   Maintenance Plan

The research group that created *Planetarium* will maintain the *Planetarium* dataset and benchmark with an open GitHub repository and issue submission system, and the dataset will be hosted on HuggingFace. The maintenance plan includes regular issue tracking, with reviews and categorization of issues, aiming to resolve high-priority issues within a week and scheduling minor updates quarterly. Comprehensive documentation and automated testing will ensure quality and compatibility.

---

[1]`hf.co/datasets/BatsResearch/planetarium`
[2]`github.com/BatsResearch/planetarium`

H   Text-to-PDDL Pair Example

BLOCKS WORLD

Ground Truth PDDL

```
(define (problem equal_towers_to_equal_towers_5)
    (:domain blocksworld)
    (:requirements :strips)
    (:objects b1 b2 b3 b4 b5)
    (:init (arm-empty)
           (clear b5)
           (on b2 b1)
           (on b3 b2)
           (on b4 b3)
           (on b5 b4)
           (on-table b1))
    (:goal (and (arm-empty)
                (on-table b1)
                (on b2 b1)
                (on b3 b2)
                (on b4 b3)
                (on b5 b4)
                (clear b5)))
)
```

Natural Language Description

(objects: You have 5 blocks, b1 through b5, stacked into 1 towers of equal heights, and your arm is empty.) (abstract init: stacked into 1 towers of equal heights, and your arm is empty. | explicit init: Your arm is empty. b1 is on the table. b2 is on b1. b3 is on b2. b4 is on b3. b5 is on b4. b5 is clear.) Your goal is to have the following: (abstract goal: stack the blocks into 1 towers of equal heights. | explicit goal: Your arm should be empty. b1 should be on the table. b2 should be on b1. b3 should be on b2. b4 should be on b3. b5 should be on b4. b5 should be clear.)