One Demo Is All It Takes: Planning Domain Derivation with LLMs from A Single Demonstration

Jinbang Huang[1,†], Yixin Xiao[1], Zhanguang Zhang[1], Mark Coates[2], Jianye Hao[1], Yingxue Zhang[1]
[1]Huawei Noah's Ark Lab, [2]McGill University
[†]jinbang.huang@h-partners.com

Abstract

Pre-trained Large Language Models (LLMs) have shown promise in solving planning problems but often struggle to ensure plan correctness, especially for long-horizon tasks. Meanwhile, traditional robotic task and motion planning (TAMP) frameworks address these challenges more reliably by combining high-level symbolic search with low-level motion planning. However, TAMP relies on the availability of planning domains that typically involve substantial manual effort and domain expertise, limiting its generalizability. We introduce Planning Domain Derivation with LLMs (PDDLLM), a novel approach that combines simulated physical interaction with LLM reasoning to improve planning performance. The method reduces reliance on humans by inferring planning domains from a single annotated task-execution demonstration. Unlike prior domain-inference methods that rely on partially predefined or language descriptions of planning domains, PDDLLM constructs domains entirely from scratch and automatically integrates them with low-level motion planning skills, enabling fully automated long-horizon planning. PDDLLM is evaluated on over 1,200 diverse tasks spanning nine environments and benchmarked against six LLM-based planning baselines, demonstrating superior planning performance, lower token costs, and successful deployment on multiple robot platforms.

1. Introduction

Robotic planning remains challenging in complex scenarios requiring abstract, long-horizon reasoning. Large language models (LLMs) show strong generalization in this domain but often struggle with temporal dependencies in long-horizon tasks. Task and Motion Planning (TAMP) frameworks offer a solution by integrating high-level symbolic reasoning with low-level motion planning, yet they rely on manually designed planning domains, abstract world models that are labor-intensive to create and limit adaptability. To overcome this, recent advances in world model learning aim to support long-horizon decision making via learned world models [14, 23, 34, 40], using either natural language or formal definitions like planning domain definition language (PDDL) [26]. Building on this paradigm, our approach leverages the concept of planning domains from TAMP to enhance the long-horizon reasoning capability of LLMs. By combining LLMs with physical simulation, we automatically generate a task-specific PDDL planning domain from a single expert demonstration and a brief task description. The resulting PDDL planning domain can be seamlessly integrated with a low-level motion planner, enabling the system to solve tasks with greater complexity than the original demonstration.

Our method is the first to construct planning domains entirely from scratch, without relying on any predefined predicates, actions, or detailed human descriptions. Furthermore, it fully automates the deployment of planning domains by automatically integrating them with low-level motion planners, eliminating the need for manual intervention. The main contributions of this paper are as follows: (1) A novel framework that integrates an automatically-generated planning domain with a low-level motion planner to address long-horizon robotic planning tasks. (2) An algorithm that combines LLMs and physical simulation to automatically generate a human-interpretable planning domain from a single human demonstration. (3) We introduce a logic-constrained action sampler (LoCAS), which automatically integrates the derived planning domain with low-level motion planning for effective task execution. (4) We evaluate PDDLLM on over 1,200 tasks across nine environments, demonstrating superior performance and more efficient token usage compared to state-of-the-art baselines. Furthermore, we successfully deploy PDDLLM on two physical robot platforms.

2. Related Work

Task planning with pre-trained large models With the advent of pre-trained large models (LMs), the use of LLMs and vision language models (VLMs) has significantly ad-

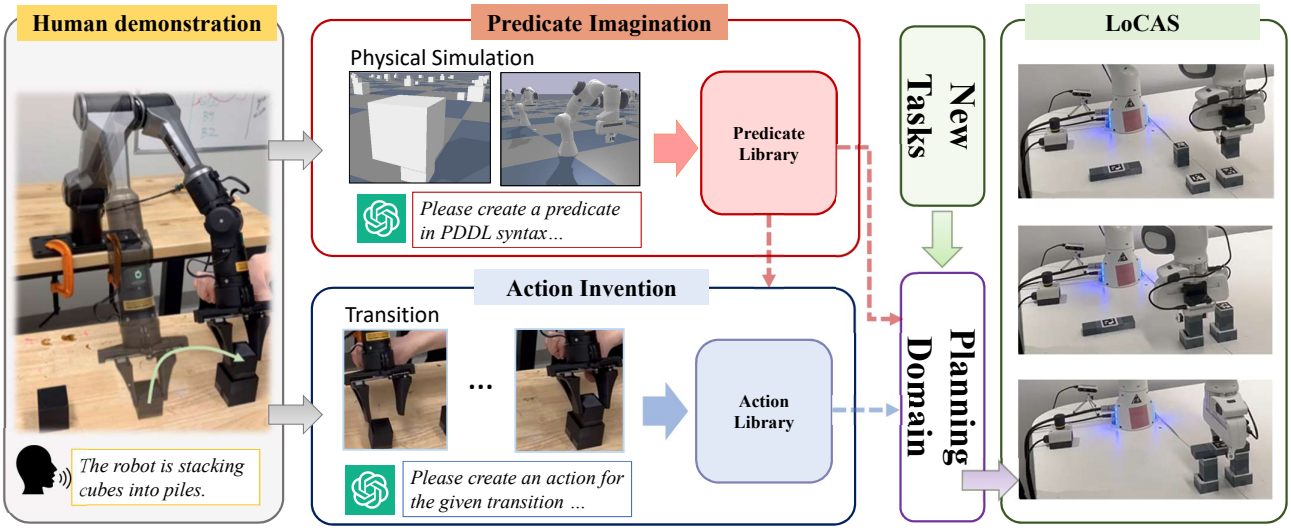

Figure 1. Overview of the proposed framework. (1) Human demonstrations, in the form of manipulation trajectories, and the corresponding task descriptions, serve as input. (2) PDDLLM initiates thousands of parallel simulations to imagine predicates by summarizing the outcomes with LLMs, returning the predicate library along with the relevance of each predicate to the current task. (3) Actions are invented by an LLM that summarizes logical state transition patterns from the demonstration, which is grounded into logical states using the imagined predicates. (4) The predicates and actions are compiled into a planning domain, which is automatically interfaced with motion planning algorithm by the Logic-Constrained Action Sampler (LoCAS) to solve new tasks.

vanced the performance of task planners [5, 18, 21, 36]. Although many studies have demonstrated the generalization capabilities of LM-based task planners, they often lack robustness and struggle with long-horizon tasks that require complex reasoning [2, 9, 17, 30, 36]. To address this limitation, recent research has explored guiding task planners with LM-derived heuristics to accelerate informed search. These approaches integrate symbolic search with LMs to accelerate task planning and reduce search complexity. Notable efforts include heuristics for prioritizing feasible states [40], ranking feasible actions [15, 27, 39, 40], and pruning search trees [33]. However, a major limitation of these methods lies in their reliance on manually constructed symbolic planning domains to build search trees, which imposes additional development overhead and reduces flexibility.

Learning planning domains A recent line of research aims to infer the planning domain directly from human demonstrations, environment interactions, or natural language. However, these approaches often depend on partially or fully predefined symbolic predicates and actions [16, 20, 25, 32, 37, 42]. Some recent studies have explored leveraging LMs for domain generation, primarily by extending manually defined domains with additional predicates and actions [1, 3, 23]. The approach proposed by Guan et al. [14] generates planning domains requiring prompts containing expert-crafted PDDL domain examples and detailed descriptions of predicates and actions. Moreover, many of these studies assume that the robot is already equipped with motion planning skills aligned with the logical actions [1, 16, 20, 23]. This assumption necessitates manual integration between the symbolic planning domain and low-level motion planners, thereby constraining scalability and limiting autonomy.

3. Preliminary

PDDL is a standard formal language used to specify planning problems. The object set $\mathcal{O}$ represents the environment's objects, whose continuous state $\mathcal{S}$, such as pose, color, and size, can be queried via a perception function $\mathcal{I} : \mathcal{O} \times \mathcal{I} \to \mathcal{S}$. The PDDL domain $\mathcal{D} = (\mathcal{P}, \mathcal{A})$ describes the general rules of the environment, consisting of a set of logical predicates $\mathcal{P}$ and a set of logical actions $\mathcal{A}$. A **logical predicate** $p \in \mathcal{P}$ specifies either intrinsic properties of an object o or relations between objects (e.g., `(cooked ?o1)`, `(on ?o1 ?o2)`). Each predicate is evaluated by a binary classifier over the continuous state, returning true or false. Grounding $\mathcal{P}$ across $\mathcal{S}$ produces a symbolic description of the environment $\mathcal{S} \times \mathcal{P} \to \mathcal{X}$. A **logical action** $a \in \mathcal{A}$ consists of a precondition $\mathcal{P}_{pre} = \langle p_1, p_2, \dots \rangle$ and an effect $\mathcal{P}_{eff} = \langle p'_1, p'_2, \dots \rangle$. The precondition represents a set of predicates that must be satisfied for the action to be executed, while the effect describes the change of the resultant state upon action completion. Logical actions define the logical state transitions $\mathcal{X}^{(t)} \times a^{(t)} \to \mathcal{X}^{(t+1)}$. Thus, any planning problem $\langle \mathcal{S}^{(init)}, \mathcal{S}^{(final)} \rangle$ can be formulated as a logical planning problem $\mathcal{Q} = \langle \mathcal{O}, \mathcal{D}, \mathcal{X}^{(init)}, \mathcal{X}^{(final)} \rangle$,

which is solved by a symbolic planner to produce a task plan $a^{(0)}, a^{(1)}, \ldots, a^{(T-1)} = \texttt{PDDLSolver}(\mathcal{Q})$. Each logical action a must then be integrated with corresponding motion planning skills to generate continuous robotic actions $\tilde{a}$ for execution.

4. Problem Statement

We address the robot planning problem given a single human demonstration τ_{demo} and its corresponding task description T_{demo}. The demonstration τ_{demo} is represented as a sequence of continuous environment states $\tau_{demo} = \{\mathcal{S}^{(0)}, \mathcal{S}^{(1)}, \ldots\}$ and T_{demo} is a brief natural language phrase. We assume the robot is the sole agent in the environment [16]. Additionally, we assume that the demonstration covers the necessary domain knowledge to solve the target task. The proposed PDDLLM framework aims to generate a sequence of continuous robotic actions for a new planning problem. The problem can be formulated as:

$$\tilde{a}^{(0)}, \ldots, \tilde{a}^{(L-1)} = \text{PDDLLM}(\mathcal{S}_{new}^{(init)}, \mathcal{S}_{new}^{(goal)}, T_{demo}, \tau_{demo}) \tag{1}$$

where $\mathcal{S}_{new}^{(init)}, \mathcal{S}_{new}^{(goal)}$ define the new planning problem and L is the plan length.

5. Methodology

Figure 1 presents an overview of the PDDLLM framework. With T_{demo} and τ_{demo} as inputs, PDDLLM constructs a relevant predicate library through predicate imagination and generates an action library via action invention. Ultimately, these predicate and action libraries are compiled into an executable PDDL planning domain, automatically interfaced with motion planners via LoCAS. In the following sections, we provide a detailed explanation of each step in the framework.

5.1. PDDL Domain Generation

Given a human demonstration and the corresponding task description, we combine simulated physical interaction and LLM reasoning to produce an executable PDDL planning domain through predicate imagination and action invention.

5.1.1. Predicate Imagination:

The process of predicate imagination consists of two stages. Stage (1) generate first-order predicates and Stage (2) futher expand the higher-order predicates.

Stage (1) In this stage, PDDLLM generates first-order predicates, which directly describe the physical properties or relations of the objects (e.g., `(is_on ?o1 ?o2)`, `(smaller ?o1 ?o2)`), by summarizing simulated object interactions using an LLM.

Definition 1 (Feature Space)**.** *The feature space is defined as a set of continuous state variables, such as position, size, and color, that fully characterize the state of each object in the environment.*

Following Definition 1, the feature space is defined as a set of variables, such as Cartesian coordinates (x, y, z) for object positions and RGB values (r, g, b) for colors. The feature space is bounded by real-world constraints. Object states are uniformly sampled across this space to span a diverse range of object-object and object-environment interactions. Each sampled state undergoes a two-step verification process. First, it is evaluated for physical feasibility using a physical simulator, which serves as a physical knowledge base to capture complex dynamics beyond the reasoning capacity of LLMs. Only physically valid states are retained.

Next, PDDLLM partitions the feature space into a finite collection of subspaces. The range of each feature is divided into intervals, with the length of each interval being a hyperparameter. The intersections of these intervals specify the subspaces. Each object state can be mapped into one of the subspaces. Each subspace is analyzed to determine whether it contains feasible object states, as verified through simulation in the previous verification step. If so, prompts are generated using a predefined template to describe the corresponding scenes. Prompt generation is automated as it only requires the replacement of some keywords (such as "position" with "color") and the specification of interval boundaries. An LLM is then prompted to summarize subspaces into meaningful predicates and select those relevant to the task. The subspace boundaries serve as predicate implications, enabling the classification of whether a predicate holds true. Figure 2.a illustrates an example of predicate generation for positional relations between objects.

Stage (2) First-order predicates are combined with logical operators and quantifiers to construct higher-order predicates. They are systematically combined in all possible ways to generate more complex logical expressions. Following prior work [7, 32], we primarily use the negation operator, along with the quantifiers "for all" ($\forall$) and "there exists" ($\exists$). For example, `(is_on ?o1, ?o2)` can be negated to produce `(not_is_on ?o1, ?o2)`. When combined with the universal quantifier, this further yields `(∀_o1_not_is_on ?o1, ?o2)`, meaning that for any o_1 in the environment, `(is_on ?o1, ?o2)` is false. This indicates that object o_2 is on top.

5.1.2. Action Invention:

After constructing the predicate library, the human demonstration τ_{demo} can be mapped into the logical space as τ_{demo}^{logic} by grounding all relevant predicates at each time step. The logical state transitions within τ_{demo}^{logic} signify the execution of actions. An advantage of learning actions in the logical space is that it simplifies pattern recognition by focusing on moments of logical state transitions, effectively transforming

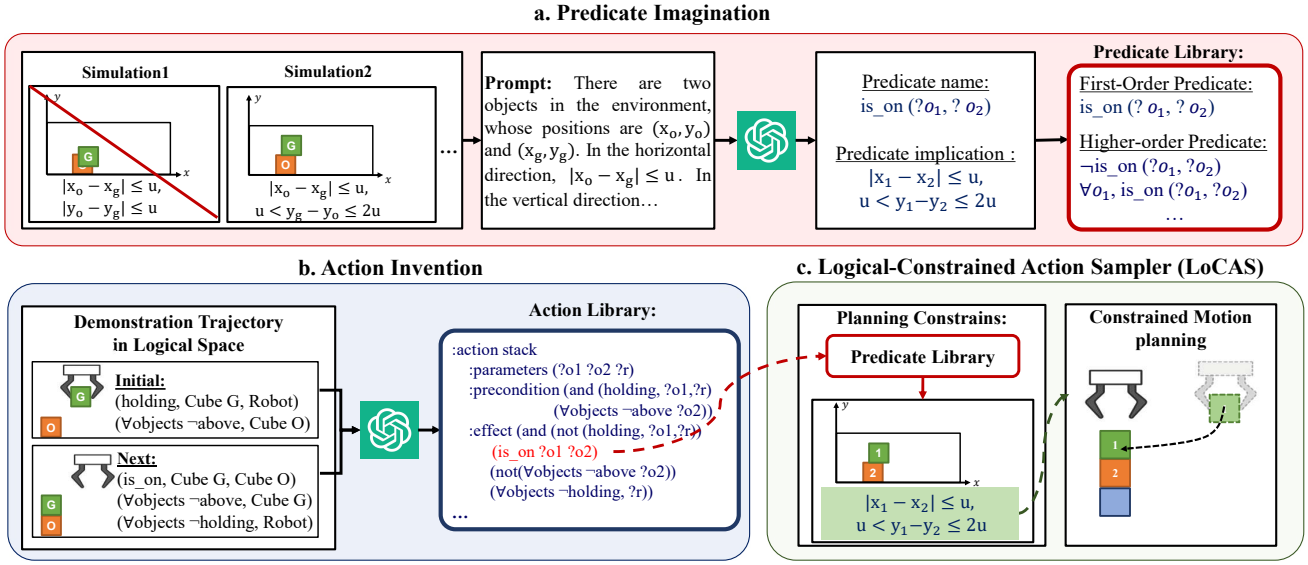

Figure 2. **a.** This example illustrates the imagination of predicates for relative object positions. Let u be a configurable variable determined by the feature subspace size. Object poses are sampled and simulated, with infeasible cases (e.g., Simulation 1) filtered out. Feasible subspaces are encoded into LLM prompts, and the LLM returns task-relevant predicates. **b.** This example shows how *Stack* actions are invented. Continuous states are grounded into logical states using the imagined predicates, where the state transition represents the logical action. By prompting the LLM with the transition from the initial state to the next state, we obtain the PDDL definition of action *Stack*. **c.** The integration of actions with the motion planner is handled automatically by LoCAS, which converts first-order predicates in the action effect $\mathcal{P}_{eff}$ into geometric constraints for motion planning.

long trajectories into concise logical representations. For instance, the continuous manipulation trajectory τ_{demo} in Figure 2.b contains over 1000 time steps while τ_{demo}^{logic} is reduced to merely 2 steps. Logical state transitions are extracted from pre-and-post-change states and presented to the LLM as prompts. After experimenting with various prompt structures, we found this direct, structured method to be the most effective. A concrete example of action invention for *Stack* is elaborated in Figure 2.b.

5.2. Logic-Constrained Action Sampler (LoCAS)

Having obtained the logical action definitions, the next step is to establish an interface between each logical action and the low-level motion planner. This is achieved through the Logic-Constrained Action Sampler (LoCAS). Traditional motion planning algorithms search for robot pose sequences within the workspace until a feasible trajectory is found [10]. Later, Toussaint [35] proposed encoding logical constraints as mathematical expressions for optimization. Combining these ideas with domain generation motivate us to develop LoCAS. It ensures that the first-order predicates in a logical action's effect $\mathcal{P}_{\text{eff}}$ are satisfied upon action completion by enforcing them as constraints during the motion planning process, as is in Figure 2.c. These predicate implications, expressed as mathematical inequalities, guide the motion planner. Consequently, each logical action is transformed into a standard constrained motion planning problem. The adoption of LoCAS automatically grounds logical actions with motion planners and thus eliminates the need for predefined skills, which are human-designed or deep-learned task-specific motion planning policies commonly assumed available in previous studies [20, 23].

Planning Pipeline: The user provides a single human demonstration τ_{demo} and its corresponding task description T_{demo}, based on which PDDLLM automatically constructs a planning domain $\mathcal{D}_{\text{PDDLLM}}$ and integrates it with low-level motion planners, enabling direct deployment in robot systems. When a novel planning task $(\mathcal{S}_{new}^{(init)}, \mathcal{S}_{new}^{(final)})$ is presented, the initial and goal states are first mapped to their corresponding logical representations using the invented predicates. For the formulated logical planning problem $\mathcal{Q}_{new} = (\mathcal{O}, \mathcal{D}_{\text{PDDLLM}}, \mathcal{X}_{new}^{(init)}, \mathcal{X}_{new}^{(final)})$, `PDDLSolver` is used to solve for the task plan, which is refined into continuous action sequence $\{\tilde{a}^{(0)}, \tilde{a}^{(1)}, \ldots, \tilde{a}^{(L-1)}\}$ by LoCAS.

6. Experiments and Baselines

Our Experiments are conducted in PyBullet Simulation [6], with the symbolic solver from PDDLStream [13] and the motion planning algorithm from the PyBullet-Planning [11, 12]. The LLM we used is GPT-4o [28].

Table 1. Maximum planning complexity and domain derivation complexity of each category .

Task	Stack	Unstack	Color Classify	Alignment	Parts Assembly	Rearrange	Burger Cook	Bridge Build	Tower of Hanoi
Max Planning Complexity	10^{58}	10^{58}	10^{71}	10^{58}	10^{58}	10^{137}	10^{48}	10^{36}	10^{307}
Domain Derivation Complexity	90	90	111	94	92	96	114	128	94

Task Diversity: To ensure robustness and broad applicability, we sampled over 1,200 planning tasks across nine distinct environments. *Stacking* involves placing objects to form stable stacks; *unstacking* requires removing items without disturbing others; *rearrangement* moves objects from an initial to a target layout; *alignment* lines up objects with uniform spacing and orientation; *color classification* groups and places objects by color; *parts assembly* involves sequentially combining mechanical components; *Tower of Hanoi* is a disk-moving puzzle; *bridge building* arranges blocks into a bridge structure; and *burger cooking* stacks ingredients to assemble burgers. Each task category includes tasks of 3 to 20 objects, with 10 distinct tasks sampled for each object count. The resulting task plan lengths ranged from 6 to 510 steps, reflecting a wide spectrum of planning horizons. Our experiments spanned over 150 unique predicates.

Task Complexity: In task design, we consider two types of complexity: domain derivation complexity and planning complexity. *Domain derivation complexity* is determined by the number of predicates imagined and actions invented by PDDLLM; the more predicates and actions, the higher the complexity. *Planning complexity* is influenced by both the planning domain and the task. Given n objects, a task plan of length l, and m actions in the domain, the branching factor at each step is $m \times n$, resulting in a approximate complexity of $(m \times n)^l$. Table 1 showcase the planning complexity (in approximate order of magnitude) of the most difficult problem in the category and the domain derivation complexity. The Tower of Hanoi task exhibits the greatest planning complexity, while bridge building presents the highest domain derivation complexity. In contrast, stack and unstack are simpler in both complexity measures.

Knowledge Transferability: We evaluate the knowledge transferability of PDDLLM by testing its ability to generalize from demonstrated tasks to novel ones with overlapping predicates and actions. For most tasks, PDDLLM was given demonstrations of the same task involving fewer (3 to 4) objects. However, for compositional tasks such as rearrangement and bridge building, the model was instead provided demonstrations of simpler subtasks. Specifically, rearrangement used demonstrations from both stacking and unstacking, while bridge building combined demonstrations of stacking and alignment. These setups test whether PDDLLM can compose skills learned from simpler demonstrations to solve more complex tasks.

6.1. Baselines and Ablations

We implement six baselines and one ablation of PDDLLM to comprehensively evaluate our method. GPT-4o is used as the default LLM unless otherwise specified, and the motion planning algorithm is kept the same for all methods.

- **LLMTAMP**, **o1-TAMP**, **R1-TAMP**: LLM-based task and motion planning (LLMTAMP) inspired by Huang et al. [17], Li et al. [22], which use LLMs for task planning, with language description of planning task as input. The task execution demonstration, same as those used in PDDLLM, was provided in the form of natural language in prompt. In addition to GPT-4o, reasoning LLMs, OpenAI's o1 [29] and Deepseek's R1 [8], are used as backbones for o1-TAMP and R1-TAMP, respectively.
- **LLMTAMP-FF**: Following the method by Chen et al. [4], Huang et al. [18], LLMTAMP-FF extends LLMTAMP with a failure feedback loop.
- **LLMTAMP-FR**: Following Wang et al. [36], LLMTAMP-FR extends failure detection by providing specific failure reasons to guide replanning with the LLM.
- **Expert Design**: The expert design baseline uses expert-crafted planning domains with symbolic solvers. Expert-designed domains are refined from PDDLLM-derived domains by an expert.
- **RuleAsMem**: In addition to the six baselines, we include an ablation of our method, RuleAsMem, which is an ablation of PDDLLM that treats the generated PDDL domain as contextual memory.

Robot planning is required to be real-time in robot deployment, imposing constraints on the planning time allowance [10, 12, 13]. In our experiment, a uniform planning time limit of 50 seconds is applied to all planning problems and methods. We measure performance using the planning success rate, as in other studies [16, 20, 32]. The planning time and token cost are used to measure the planning cost [41]. Three parallel runs were conducted to compute the mean and standard error of the means for the planning success rate.

Table 2. Planning success rate (%) across tasks for all methods (time limit = 50 s). The best results are highlighted in bold. `Expert` is excluded from the comparison, as it requires additional manual effort and serves as an upper bound.

Method	Expert	LLMTAMP	LLMTAMP-FF	LLMTAMP-FR	RuleAsMem	PDDLLM
Stack	96.1 ± 0.2	41.7 ± 4.3	70.8 ± 1.4	64.2 ± 3.1	85.5 ± 2.9	$\mathbf{97.5 \pm 1.6}$
Unstack	96.1 ± 0.2	89.4 ± 1.5	94.6 ± 0.9	92.1 ± 2.3	88.4 ± 1.2	$\mathbf{97.7 \pm 0.7}$
Color Classification	100 ± 0.0	18.1 ± 1.5	36.4 ± 1.1	49.0 ± 3.0	88.7 ± 2.3	$\mathbf{100 \pm 0.0}$
Alignment	100 ± 0.0	31.1 ± 3.1	52.0 ± 2.7	40.0 ± 2.4	96.0 ± 0.8	$\mathbf{100 \pm 0.0}$
Parts Assembly	98.9 ± 0.6	33.3 ± 1.5	53.9 ± 1.1	41.3 ± 1.2	95.0 ± 0.6	$\mathbf{100 \pm 0.0}$
Rearrange	63.7 ± 1.3	5.6 ± 1.0	17.4 ± 1.1	11.8 ± 1.8	1.1 ± 0.6	$\mathbf{64.3 \pm 0.7}$
Burger Cooking	100 ± 0.0	27.8 ± 2.8	50.0 ± 4.8	48.6 ± 6.9	27.8 ± 2.8	$\mathbf{91.7 \pm 4.8}$
Bridge Building	100 ± 0.0	43.3 ± 3.3	53.3 ± 3.8	51.7 ± 2.5	20.0 ± 0.0	$\mathbf{87.2 \pm 4.3}$
Tower of Hanoi	100 ± 0.0	14.3 ± 0.0	14.3 ± 0.0	14.3 ± 0.0	14.3 ± 0.0	$\mathbf{100 \pm 0.0}$
Overall	93.4 ± 0.1	35.7 ± 0.5	52.5 ± 0.4	48.6 ± 0.8	69.9 ± 0.7	$\mathbf{93.3 \pm 0.7}$

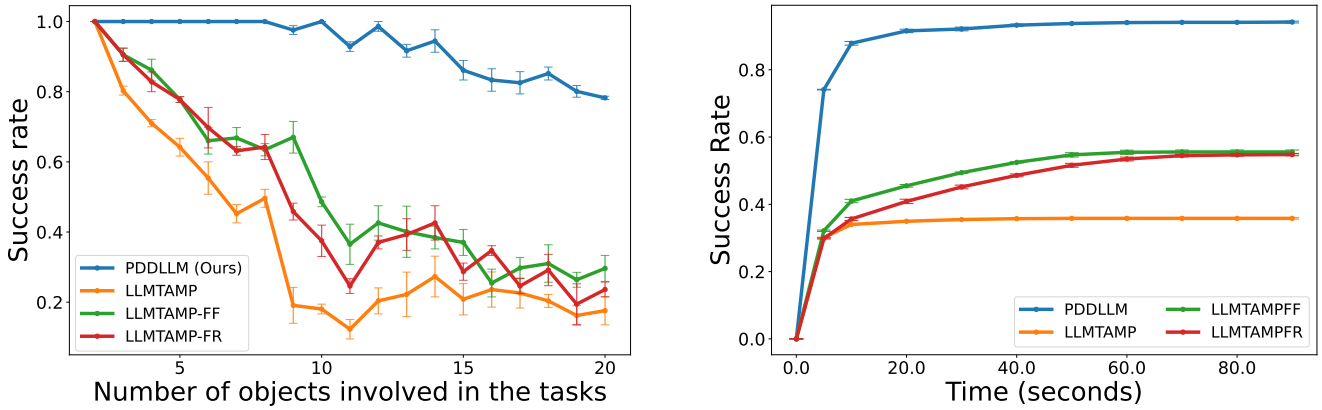

Figure 3. (left) Planning success rate trend across increasing object counts. (right) Overall planning success rate under varying time limits.

7. Results

Through the experiments, we aim to answer the following research questions: (1) How does PDDLLM perform relative to other LLM-based planners? (2) Can PDDLLM generalize to unseen, more complex tasks? (3) Does PDDLLM derive high-quality domains with performance comparable to expert designs? (4) How does PDDLLM's token cost compare to other LLM-based planners?

Q1. Performance comparison to baselines: Table 2 presents the planning success rate of all evaluated methods across all tasks, measured with a 50-second time limit. PDDLLM shows clear advantages in planning efficiency and generalizability over baseline methods. While LLM-based baselines perform competitively in simpler tasks like stacking and unstacking, their performance drops sharply in complex tasks such as rearrangement, burger cooking, bridge building, and Tower of Hanoi. In contrast, PDDLLM maintains strong performance across all task categories, achieving an over 40% improvement in overall planning success rate compared to the best LLM-based planner baseline (i.e., LLMTAMP-FF). Even the ablated variant of PDDLLM, RuleAsMem, outperforms the LLM-based planners by at least 17.4% in overall success rate. Compared to PDDLLM, RuleAsMem exhibits less stability. It performs well in simpler tasks but struggles in more complex ones, suggesting the LLMs struggle to understand complex domains. In addition to the results evaluated with a 50-second time window, we also report the overall success rates of the main methods across varying time limits. As shown in Figure 3(right), PDDLLM consistently outperforms the baseline methods and demonstrates superior planning performance across all time limits. Notably, it reaches performance saturation the fastest, highlighting its superior time efficiency among all evaluated approaches.

We further compare PDDLLM's planning ability with more powerful reasoning LLMs (OpenAI-o1 and DeepSeek-R1) in Table 3. While reasoning-based models exhibit strong

Table 3. Comparison of planning success rate (%) and token cost (k) between PDDLLM and LLMTAMP and the reasoning LLM variants. The best results are shown in bold, and the second-best results are underlined.

Task	Success Rate (%) ↑				Token Cost (k) ↓			
	PDDLLM	LLMTAMP	o1-TAMP	R1-TAMP	PDDLLM	LLMTAMP	o1-TAMP	R1-TAMP
Rearrangement	$\mathbf{73.8 \pm 1.1}$	5.6 ± 1.0	$\underline{70.8 \pm 1.5}$	40.0 ± 5.0	$\underline{334}$	**212**	1200	1460
Tower of Hanoi	$\mathbf{100.0 \pm 0.0}$	14.3 ± 0.0	$\underline{33.3 \pm 2.4}$	14.3 ± 0.0	535	**36**	529	$\underline{353}$
Bridge Building	$\mathbf{87.2 \pm 4.3}$	44.3 ± 3.3	$\underline{51.7 \pm 2.5}$	40.0 ± 0.0	375	**50**	$\underline{270}$	363
Overall	$\mathbf{80.5 \pm 0.5}$	13.9 ± 0.9	$\underline{61.5 \pm 1.3}$	35.9 ± 3.1	$\underline{415}$	**99**	666	725

Table 4. Percentage of missing or redundant predicates and actions across selected domains.

Task	Stack	Burger Cooking	Bridge Building	Tower of Hanoi
Percentage of missing predicates or actions	0.0%	18.8%	16.7%	0.0%
Percentage of redundant predicates or actions	10.0%	0.0%	0.0%	11.1%

planning capabilities, their high computational cost prevents them from completing within the 50-second window. To sufficiently compare the planning capabilities, we extend the time limit to 500 seconds and evaluate them on the most challenging tasks. Although o1-TAMP and R1-TAMP show remarkable improvement compared to GPT-4o-based LLMTAMP, they remain less robust in long-horizon planning and fail to match PDDLLM's performance. In contrast, our method, relying solely on GPT-4o, consistently achieves higher success rates across complex tasks.

Q2. Generalization: The experimental results highlight PDDLLM's ability to handle increasing complexity in both planning and domain derivation, consistently outperforming baseline methods. In terms of planning complexity, as shown in Figure 3(left), PDDLLM maintains robust planning performance as task complexity grows, achieving high success rates even in scenarios involving up to 20 objects. As shown in Table 2, performance degradation is observed in more challenging tasks such as rearrangement, where longer action sequences are required and there are more complex motion constraints. From the perspective of domain derivation complexity, PDDLLM remains effective even in tasks demanding the generation of over 100 predicates. However, success rates drop in the most complex domains, such as bridge building, primarily due to missing supporting predicates. Full discussion can be found in Section 9

PDDLLM demonstrates a strong ability to integrate knowledge across demonstrations. The modular nature of PDDL action syntax allow the easy transfer of actions learned from different demonstrations among domains. Despite receiving only one example each for stacking and unstacking, it successfully combines the "stack" action and the "unstack" action to solve rearrangement tasks. Similarly, in the bridge building domain, PDDLLM successfully combines the "align" action and the "stack" action using one demonstration of cube alignment and one of stacking. The high success rate of these tasks in Table 2 underscore the robustness of the derived planning domains.

Q3. Domain Quality: We evaluate the quality of planning domains generated by PDDLLM by comparing the percentage of missing or redundant predicates and actions, as well as the planning success rate across a range of tasks, against expert-designed domains. Table 4 reports the percentage of missing or redundant elements in the final domains; tasks with no such issues are omitted. These results indicate that PDDLLM produces high-quality domains with minimal errors, even in complex scenarios. While a few predicates may be absent, the overall logical structure remains sound, as reflected in the consistently high planning success. As shown in Table 2, the derived domains achieve 93.3% success rate, closely matching the performance of expert-crafted domains.

Q4. Token efficiency: We compare the token efficiency of our method against LLM-based baselines on the three most complex tasks, as shown in Table 3. Although the GPT-4o-based LLMTAMP incurs lower total token costs, it performs poorly across all three tasks. Compared to o1-TAMP and R1-TAMP, our PDDLLM uses significantly fewer tokens while consistently achieving better performance. These results also underscore the challenge of deploying reasoning LLMs on real robot systems, given their high token consumption and time costs in planning.

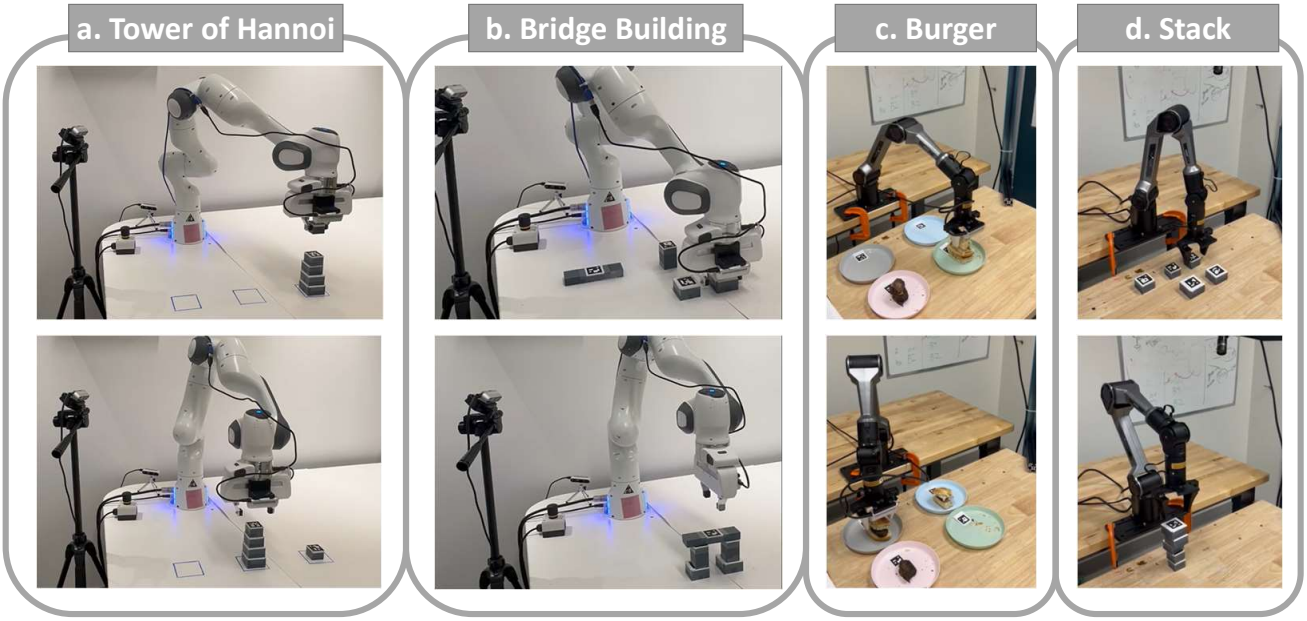

Figure 4. Real robot experiment with Agilex Piper and Franka Panda Arm: **a.** Franka Panda solving the tower of Hannoi puzzle. **b.** Franka Panda building a bridge. **c.** Piper arm making burgers. **d.** Piper arm stacking cubes.

8. Real Robot Deployment

We evaluate PDDLLM's validity on real-world robotic systems to demonstrate its cross-platform deployability. The system is tested on both the Agilex Piper Arm and the Franka Panda Arm. ArUco 5×5 markers are used for pose estimation, while ROS2 and MoveIt2 serve as the robot driver and motion planning framework. Both robots successfully complete real-world planning tasks through the direct deployment of PDDLLM, demonstrating its effectiveness across different hardware platforms and validating its applicability in real-world robotic scenarios. As indicated in Figure 4, four real robot experiments, including stacking, bridge building, cooking burgers, and the tower of Hannoi are performed.

9. Limitation

A current limitation of PDDLLM lies in its occasional omission of logically complex predicates, which contributes to the observed decline in success rates for more complex domains such as Burger Cooking and Bridge Building, as shown in Table 2. For example, in the bridge building task, the robot must assemble the top surface only after all base components are correctly placed. If the surface is assembled prematurely, it becomes difficult to generate feasible manipulation trajectories for the remaining base components, often resulting in planning failure. In contrast, the expert-designed planning domain addresses this issue by introducing a predicate `(all_base_finished)` as a precondition for surface assembly, thereby enforcing the correct ordering of actions. Recent research has begun to address this limitation, for example, by leveraging VLMs to invent new predicates [23], or by refining the domain through environmental feedback [16].

10. Conclusion

This paper presents PDDLLM, the first approach in the field to generate a complete planning domain from scratch, without relying on any predefined predicates or actions. By extracting logical structures directly from pre-trained LLMs, PDDLLM autonomously derives both predicates and actions, enabling fully automated domain construction. Evaluated across a wide range of environments, PDDLLM demonstrates high quality in domain derivation and strong generalizability across diverse task categories. Moreover, when integrated with the LoCAS framework, PDDLLM fully automates the integration between the PDDL planning domain and the low-level motion planner. This level of automation significantly improves usability and positions the framework as an adaptable and scalable solution for robotic planning and decision-making. Compared to existing methods, PDDLLM outperforms other LLM-based baselines and closely matches the performance of expert-designed planning domains, particularly in complex and long-horizon planning scenarios. Future work will focus on integrating perception into the system, enabling domain derivation from raw sensory inputs. From the aspect of broader impact, PDDLLM lowers the barrier to deploying robots by enabling domain derivation from demonstrations, making automation more accessible.

References

[1] Ashay Athalye, Nishanth Kumar, Tom Silver, Yichao Liang, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Predicate invention from pixels via pretrained vision-language models. arXiv [cs.RO], 2024. 2

[2] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In Proc. Conf. on Robot Learning, 2023. 2

[3] Walker Byrnes, Miroslav Bogdanovic, Avi Balakirsky, Stephen Balakirsky, and Animesh Garg. CLIMB: Language-guided continual learning for task planning with iterative model building. arXiv [cs.RO], 2024. 2

[4] Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas A. Roy, and Chuchu Fan. Autotamp: Autoregressive task and motion planning with llms as translators and checkers. In Proc. IEEE Int. Conf. on Robotics and Automation, 2023. 5, 12

[5] Yongchao Chen, Jacob Arkin, Yilun Hao, Yang Zhang, Nicholas Roy, and Chuchu Fan. PRompt optimization in multi-step tasks (PROMST): Integrating human feedback and heuristic-based sampling. In Proc. Conf. Empirical Methods in Natural Language Processing, 2024. 2

[6] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021. 4

[7] Aidan Curtis, Tom Silver, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Kaelbling. Discovering state and action abstractions for generalized task and motion planning. In Proc. AAAI Conf. on Artificial Intelligence, 2022. 3

[8] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv [cs.CL], 2025. 5, 12

[9] Danny Driess, F Xia, Mehdi S M Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Q Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, P Sermanet, Daniel Duckworth, S Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Peter R Florence. PaLM-E: An embodied multimodal language model. In Proc. Int. Conf. on Machine Learning, 2023. 2

[10] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In Proc. IEEE Int. Conf. on Robotics and Automation, 2015. 4, 5

[11] Caelan Reed Garrett. Pybullet planning. https://pypi.org/project/pybullet-planning/, 2018. 4

[12] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Backward-forward search for manipulation planning. In Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2015. 4, 5

[13] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. Proc. Int. Conf. Autom. Plan. Sched., 2020. 4, 5

[14] Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. In Proc. Adv. Neural Inf. Proc. Systems, 2023. 1, 2

[15] Mengkang Hu, Yao Mu, Xinmiao Yu, Mingyu Ding, Shiguang Wu, Wenqi Shao, Qiguang Chen, Bin Wang, Yu Qiao, and Ping Luo. Tree-planner: Efficient close-loop task planning with large language models. arXiv [cs.CL], 2023. 2

[16] Jinbang Huang, Allen Tao, Rozilyn Marco, Miroslav Bogdanovic, Jonathan Kelly, and Florian Shkurti. Automated planning domain inference for task and motion planning. arXiv [cs.RO], 2024. 2, 3, 5, 8, 17

[17] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In Proc. Int. Conf. on Machine Learning, 2022. 2, 5, 12

[18] 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. Inner monologue: Embodied reasoning through planning with language models. In Proc. Conf. on Robot Learning, 2023. 2, 5, 12

[19] Mohamed Khodeir, Ben Agro, and Florian Shkurti. Learning to search in task and motion planning with streams. IEEE Robotics and Automation Letters, 2023. 17

[20] Nishanth Kumar, Willie McClinton, Rohan Chitnis, Tom Silver, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Learning efficient abstract planning models that choose what to predict. In Proc. Conf. on Robot Learning, 2023. 2, 4, 5

[21] Boyi Li, Philipp Wu, Pieter Abbeel, and Jitendra Malik. Interactive task planning with language models. In Proc. 2nd Workshop on Language and Robot Learning, 2023. 2

[22] Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, Jacob Andreas, Igor Mordatch, Antonio Torralba, and Yuke Zhu. Pre-trained language models for interactive decision-making. In Proc. Adv. Neural Inf. Proc. Systems, 2022. 5, 12

[23] Yichao Liang, Nishanth Kumar, Hao Tang, Adrian Weller, Joshua B Tenenbaum, Tom Silver, João F Henriques, and Kevin Ellis. VisualPredicator: Learning abstract world models with neuro-symbolic predicates for robot planning. arXiv [cs.AI], 2024. 1, 2, 4, 8

[24] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering large language models with optimal planning proficiency. arXiv [cs.AI], 2023. 12

[25] Weiyu Liu, Neil Nie, Ruohan Zhang, Jiayuan Mao, and Jiajun Wu. Learning compositional behaviors from demonstration and language. In Proceedings of The 8th Conference on Robot Learning, 2025. 2

[26] Drew McDermott, Malik Ghallab, Adele E. Howe, Craig A. Knoblock, Ashwin Ram, Manuela M. Veloso, Daniel S. Weld, and David E. Wilkins. Pddl-the planning domain definition language. 1998. 1

[27] Silin Meng, Yiwei Wang, Cheng-Fu Yang, Nanyun Peng, and Kai-Wei Chang. LLM-a*: Large language model enhanced incremental heuristic search on path planning. In Findings of the Assoc. for Comput. Linguistics: EMNLP 2024, 2024. 2

[28] OpenAI. GPT-4 technical report. arXiv [cs.CL], 2023. 4

[29] OpenAI. OpenAI o1 system card. arXiv [cs.AI], 2024. 5, 12

[30] Pierre Sermanet, Tianli Ding, Jeffrey Zhao, Fei Xia, Debidatta Dwibedi, Keerthana Gopalakrishnan, Christine Chan, Gabriel Dulac-Arnold, Sharath Maddineni, Nikhil J. Joshi, Pete Florence, Wei Han, Robert Baruch, Yao Lu, Suvir Mirchandani, Peng Xu, Pannag R. Sanketi, Karol Hausman, Izhak Shafran, Brian Ichter, and Yuan Cao. Robovqa: Multimodal long-horizon reasoning for robotics. Proc. IEEE Int. Conf. on Robotics and Automation, 2023. 2

[31] Tom Silver, Rohan Chitnis, Aidan Curtis, Joshua B. Tenenbaum, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Planning with learned object importance in large problem instances using graph neural networks. In Proc. AAAI Conf. on Artificial Intelligence, 2020. 17

[32] Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Kaelbling, and Joshua B Tenenbaum. Predicate invention for bilevel planning. In Proc. AAAI Conf. on Artificial Intelligence, 2023. 2, 3, 5

[33] Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B Tenenbaum, Leslie Kaelbling, and Michael Katz. Generalized planning in pddl domains with pretrained large language models. In Proc. AAAI Conf. on Artificial Intelligence, 2024. 2

[34] Hao Tang, Darren Key, and Kevin Ellis. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment. Proc. Adv. Neural Inf. Proc. Systems, 2024. 1

[35] Marc Toussaint. Logic-geometric programming: an optimization-based approach to combined task and motion planning. In Proc. Int. Joint Conf. on Artificial Intelligence, 2015. 4

[36] Shu Wang, Muzhi Han, Ziyuan Jiao, Zeyu Zhang, Yingnian Wu, Song-Chun Zhu, and Hangxin Liu. Llm3: Large language model-based task and motion planning with motion failure reasoning. In Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2024. 2, 5

[37] Lionel Wong, Jiayuan Mao, Pratyusha Sharma, Zachary S Siegel, Jiahai Feng, Noa Korneev, Joshua B Tenenbaum, and Jacob Andreas. Learning adaptive planning representations with natural language guidance. arXiv [cs.AI], 2023. 2

[38] Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models. arXiv [cs.CL], 2023. 12

[39] Zhutian Yang, Caelan Reed Garrett, Dieter Fox, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Guiding long-horizon task and motion planning with vision language models. In Proc. 2nd CoRL Workshop on Learning Effective Abstractions for Planning, 2024. 2

[40] Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. In Proc. Adv. Neural Inf. Proc. Systems, 2024. 1, 2

[41] Tianyang Zhong, Zhengliang Liu, Yi Pan, et al. Evaluation of openai o1: Opportunities and challenges of agi. arXiv [cs.CL], 2024. 5, 12

[42] Wang Zhu, Ishika Singh, Robin Jia, and Jesse Thomason. Language models can infer action semantics for symbolic planners from environment feedback. arXiv [cs.AI], 2024. 2

A. Appendix

A.1. Real Robot Experiment

A.1.1. Franka Panda Arm

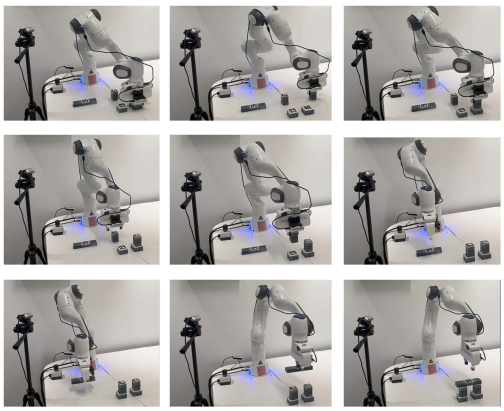

Figure 5. Franka Panda Arm building a bridge

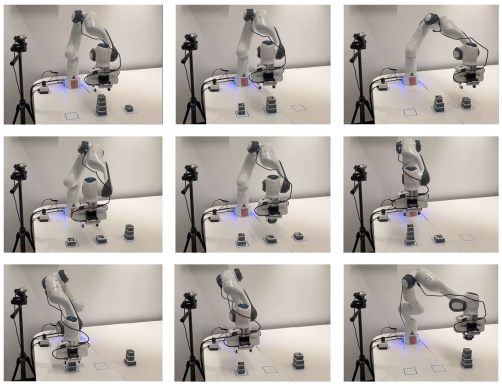

Figure 6. Franka Panda Arm solving the Tower of Hannoi puzzle

A.1.2. Agilex Piper Arm

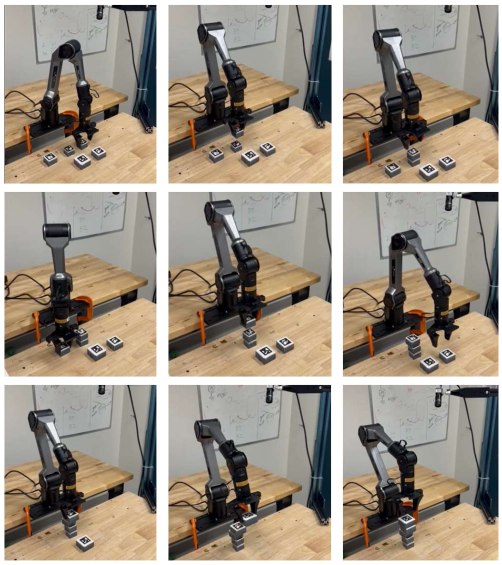

Figure 7. Agilex Piper Arm stacking cubes

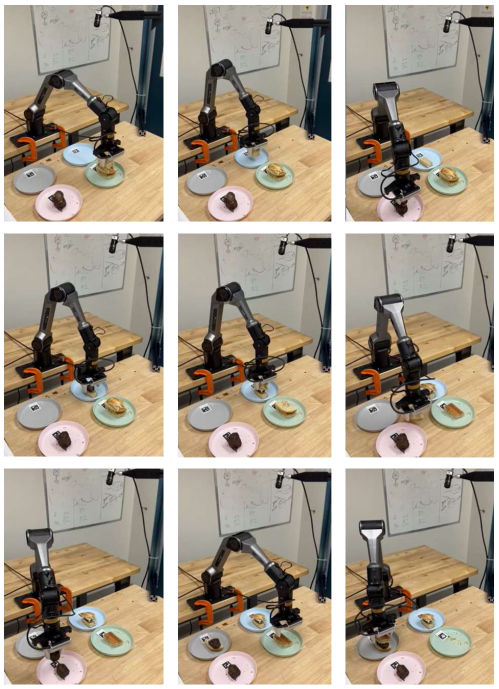

Figure 8. Agilex Piper Arm making burgers

A.2. Baseline Implementation Details

LLMTAMP: LLM-based task and motion planning (LLM-TAMP) builds on methods from [17, 22], which use pre-trained LLMs for task planning. We formulate the planning problem as a natural language description. Human demonstrations are interpreted as action sequences required to accomplish the task. The LLM then generates high-level actions to achieve the goal, which are refined into motion plans using predefined skills.

LLMTAMP-FF: LLMTAMP-FF, following the method by [4, 18], extends LLMTAMP with a failure feedback loop. Upon execution failure, the system feeds the failure signal to the LLM to regenerate the task plan, repeating until success or the time limit is reached.

LLMTAMP-FR: Following **?**], LLMTAMP-FR extends failure detection by providing specific failure reasons to guide replanning with the LLM. We design a reasoner that generates detailed explanations for plan failures and incorporates them into the prompt as feedback. The LLM performs failure reasoning and then regenerates the plan accordingly.

Expert Design: The expert design baseline uses expert-crafted planning domains to evaluate how closely PDDLLM-generated domains approach human-level performance. To highlight the readability and customizability of PDDLLM, these expert-designed domains are initialized with PDDLLM-generated outputs, which are then analyzed and refined by a TAMP expert into ground-truth domains.

o1-TAMP and R1-TAMP: As reasoning models have demonstrated superior performance in many tasks [41], we ablate the LLM in LLMTAMP to compare the planning performance of state-of-the-art reasoning LLMs with our method. Specifically, we evaluate OpenAI's o1 [29] and Deepseek's R1 [8] as the reasoning backbones.

RuleAsMem: RuleAsMem is an ablation of PDDLLM that treats the generated PDDL domain as contextual memory, rather than using it with a symbolic planner. While prior work focuses on translating language into logical representations [24, 38], RuleAsMem directly integrates logical planning rules into the LLM prompt to solve new tasks. Each task is defined by initial and goal states, using the imagined predicates, along with a human demonstration in the form of a PDDL task plan as prompts.

A.3. Prompt template

A.3.1. Predicate imagination

Template

There are n objects in the environment, whose feature name are feature value 1 and feature value 2. In dimension 1, we know feature subspace range in dimension 1. In dimension 2, we know feature subspace range in dimension 2.Please create a predicate in PDDL syntax to describe this relation and classify if it is related to the current task description. Please return the result in the following format: predicate, relevance.

Example of object position relation

Prompt: There are two objects in the environment, whose positions are (x_1, y_1) and (x_2, y_2). In the horizontal direction, $|x_1 - x_2| \leq u$. In the vertical direction, $u < y_1 - y_2 \leq 2u$. Please create a predicate in PDDL syntax to describe this relation and classify if it is related to the task of stacking cubes together. Please return the result in the following format: predicate, relevance.
LLM: is_on($?o_1$, $?o_2$), related.

Example of color relation

Prompt: There are two objects in the environment, whose colors are (r_1, g_1, b_1) and (r_2, g_2, b_2). In the red channel, $|r_1 - r_2| \leq u$. In the green channel, $|g_1 - g_2| \leq u$. In the blue channel, $|b_1 - b_2| \leq u$. Please create a predicate in PDDL syntax to describe this relation and classify if it is related to the task of stacking cubes together. Please return the result in the following format: predicate, relevance.
LLM: similar_color($?o_1$, $?o_2$), unrelated.

Example of size relation

Prompt: There are two objects in the environment, whose size are (w_1, l_1, h_1) and (w_2, l_2, h_2). In the width, $0 < w_1 - w_2 \leq u$. In the length, $0 < l_1 - l_2 \leq u$. In the height, $0 < h_1 - h_2 \leq u$. Please create a predicate in PDDL syntax to describe this relation and classify if it is related to the task of stacking cubes together. Please return the result in the following format: predicate, relevance.
LLM: larger_than($?o_1$, $?o_2$), related.

Note: Here, u is a variable determined by feature subspace range length, which depends on the user input.

A.3.2. Action invention

Template

Current state: The logical state before action execution
Next state:The logical state after the action execution
Can you give this transition an action name to summarize and describe what happened. Please follow Planning Domain Definition Language syntax.

Example for inventing 'Pick'

Prompt: The current state of the environment is:
(∀objects ¬above, o1), (is_on, o1 , o2), (∀objects ¬holding, robot)
After one action execution, the resulting state of the environment is:
(holding, o1, robot), (∀objects ¬above, o2)
Can you name this logical action to summarize and describe what happened? Please follow Planning Domain Definition Language syntax and return together with action parameters, preconditions, and effects.
LLM:

```
:action pick
    :parameters (?o1 ?o2 ?r)
    :precondition (and (is_on ?o1 ?o2) (∀objects ¬above ?o1) (∀objects ¬holding, ?r))
    :effect (and (not (is_on ?o1 ?o2)) (not (∀objects ¬holding, ?r) (∀objects ¬above, ?o2)
            (holding, ?o1,?r) (not (∀objects ¬above, o1)))
```

Example for inventing 'Place'

Prompt: The current state of the environment is:
(holding, o1, robot) (∀objects ¬above, o2)
After one action execution, the resulting state of the environment is:
(∀objects ¬above, o1) (∀objects ¬above, o2) (∀objects ¬holding, robot)
Can you name this logical action to summarize and describe what happened? Please follow Planning Domain Definition Language syntax and return together with action parameters, preconditions, and effects.
LLM:

```
:action place
    :parameters (?o1 ?r)
    :precondition (and (holding, ?o1,?r))
    :effect (and (not (holding, ?o1,?r)) (∀objects ¬above ?o1) (∀objects ¬holding, ?r))
```

A.3.3. LLMTAMP baseline

Part 1. Initialization:

Imagine you are a robot arm operator; you need to generate a sequence of actions to achieve the given goal. Here are the logical actions you can choose from: list of actions to choose from

Part 2. Human demonstrations:

Here are some examples for you to learn:
example of input and output of the system

Part 3. New planning problem:

Now you are given a new input planning problem as the following: The initial state and the goal of the problem to be solved Choose a sequence of actions to accomplish this task, and return the action sequence following the example output provided.

We integrate parts 1, 2, and 3 into a complete prompt for the LLM to generate task plans.

Example for a stacking problem

Prompt: Imagine you are a robot arm operator; you need to generate a sequence of actions to achieve the given goal. Here are the logical actions you can choose from: stack(upper box, lower box, robot), pick(upper box, table, robot).

Here are some examples for you to learn:
Example input: Object 0 is a robot. Object 1 is a table. Object 2 is a box. Object 3 is a box. Object 4 is a box. Initially, the robot is not holding anything. Object 2 is on table. Object 3 is on table. Object 4 is on table. Object 2 is the topmost object. Object 3 is the topmost object. Object 4 is the topmost object. In the goal, Object 2 is above Object 3. Object 3 is above object 4. Object 4 is on Object 1. Object 2 is the topmost object. The robot is not holding anything.
Example Output: pick(3, 1, 0), stack(3, 4, 0), pick(2, 1, 0), stack(2, 3, 0).

Now you are given a new planning problem as the following: Object 0 is a robot. Object 1 is a table. Object 2 is a box. Object 3 is a box. Object 4 is a box. Initially, Object 2 is on the table. Object 3 is on the table. Object 4 is on the table. Object 2 is the topmost object. Object 3 is the topmost object. Object 4 is the topmost object. The robot is not holding anything. In the goal, Object 2 is above object 3. Object 4 is above object 2. Object 3 is on Object 1. Object 4 is the topmost object. The robot is not holding anything. Choose a sequence of actions to accomplish this task, and return the action sequence following the example output provided.

LLM: pick(2, 1, 0), stack(2, 3, 0), pick(4, 1, 0), stack(4, 2, 0).

A.3.4. LLMTAMP+Failure Feedback baseline

LLMTAMP+Failure Feedback extends LLMTAMP with a failure feedback loop. Upon execution failure, the system feeds the failure signal to the LLM for replanning. Thus, the initial prompts of this baseline, part 1, 2, and 3, are the same as the LLMTAMP prompt. However, there is a failure summarization. Integrating part 1-4 gives the full prompt.

Part 4. Failure feedback:

Your plan failed in execution, please generate a different one. Only return the sequence of logical actions following the format of example output.

Example for a stacking problem

Prompt: Imagine you are a robot arm operator; you need to generate a sequence of actions to achieve the given goal. Here are the logical actions you can choose from: stack(upper box, lower box, robot), pick(upper box, table, robot).

Here are some examples for you to learn:
Example input: Object 0 is a robot. Object 1 is a table. Object 2 is a box. Object 3 is a box. Object 4 is a box. Initially, the robot is not holding anything. Object 2 is on table. Object 3 is on table. Object 4 is on table. Object 2 is the topmost object. Object 3 is the topmost object. Object 4 is the topmost object. In the goal, Object 2 is above Object 3. Object 3 is above object 4. Object 4 is on Object 1. Object 2 is the topmost object. The robot is not holding anything.
Example Output: pick(3, 1, 0), stack(3, 4, 0), pick(2, 1, 0), stack(2, 3, 0).

Now you are given a new planning problem as the following: Object 0 is a robot. Object 1 is a table. Object 2 is a box. Object 3 is a box. Object 4 is a box. Initially, Object 2 is on the table. Object 3 is on the table. Object 4 is on the table. Object 2 is the topmost object. Object 3 is the topmost object. Object 4 is the topmost object. The robot is not holding anything. In the goal, Object 2 is above object 3. Object 4 is above object 2. Object 3 is on Object 1. Object 4 is the topmost object. The robot is not holding anything. Choose a sequence of actions to accomplish this task, and return the action sequence following the example output provided.

LLM: pick(4, 1, 0), stack(4, 2, 0), pick(2, 1, 0), stack(2, 3, 0).

Prompt: Your plan failed in execution, please generate a different one. Only return the sequence of logical actions following the format of example output.

LLM: pick(2, 1, 0), stack(2, 3, 0), pick(4, 1, 0), stack(4, 2, 0).

A.3.5. LLMTAMP+Failure Reasoning baseline

LLMTAMP + Failure Reasoning further extends failure detection by providing specific failure reasons to guide replanning with the LLM. Part 1, 2, and 3 are the same as the LLMTAMP prompt. Integrating part 1-5 gives the full prompt.

Part 4. Failure Reasoning:

Your plan failed in execution, please generate a different one. This may involve sample new plans or reorder the last plan. Please generate output step-by-step, which includes your reasoning for the failure of the last plan. Answer the questions: (i) what is the cause of the failure of the last plan? (ii) do you see similar mistakes in other steps in the plan? Here are the failure reasons: failure reasons

Part 5. Replan:

Now, based on your above failure reasoning for (i) and (ii) , generate a new task plan to solve the original planning problem. Only return the sequence of logical actions following the format of example output.

Example for a stacking problem

Prompt: Imagine you are a robot arm operator; you need to generate a sequence of actions to achieve the given goal. Here are the logical actions you can choose from: stack(upper box, lower box, robot), pick(upper box, table, robot).

Here are some examples for you to learn:
Example input: Object 0 is a robot. Object 1 is a table. Object 2 is a box. Object 3 is a box. Object 4 is a box. Initially, the robot is not holding anything. Object 2 is on table. Object 3 is on table. Object 4 is on table. Object 2 is the topmost object. Object 3 is the topmost object. Object 4 is the topmost object. In the goal, Object 2 is above Object 3. Object 3 is above object 4. Object 4 is on Object 1. Object 2 is the topmost object. The robot is not holding anything.
Example Output: pick(3, 1, 0), stack(3, 4, 0), pick(2, 1, 0), stack(2, 3, 0).

Now you are given a new planning problem as the following: Object 0 is a robot. Object 1 is a table. Object 2 is a box. Object 3 is a box. Object 4 is a box. Initially, Object 2 is on the table. Object 3 is on the table. Object 4 is on the table. Object 2 is the topmost object. Object 3 is the topmost object. Object 4 is the topmost object. The robot is not holding anything. In the goal, Object 2 is above object 3. Object 4 is above object 2. Object 3 is on Object 1. Object 4 is the topmost object. The robot is not holding anything. Choose a sequence of actions to accomplish this task, and return the action sequence following the example output provided.

LLM: pick(4, 1, 0), stack(4, 2, 0), pick(2, 1, 0), stack(2, 3, 0).

Prompt: Your plan failed in execution, please generate a different one. This may involve sample new plans or reorder the last plan. Please generate output step-by-step, which includes your reasoning for the failure of the last plan. Answer the questions: (i) what is the cause of the failure of the last plan? (ii) do you see similar mistakes in other steps in the plan? Here are the failure reasons: Motion planning failed in pick(2, 1, 0) because of collision with Object 4.

LLM: Got it! Here are my analysis to the planning failures…..

Prompt: Now, based on your above failure reasoning for (i) and (ii) , generate a new task plan to solve the original planning problem. Only return the sequence of logical actions following the format of example output.

LLM: pick(2, 1, 0), stack(2, 3, 0), pick(4, 1, 0), stack(4, 2, 0).

A.3.6. RuleAsMem

The overall template of RuleAsMem is very similar to that of LLMTAMP. However, in RuleAsMem, the planning domain is provided and the problem is defined in PDDL syntax.

Part 1. Initialization:

Imagine you are a robot arm operator, you need to generate a sequence of actions to achieve the given goal. Here is the PDDL planning domain: PDDL planning domain

Part 2. Human demonstrations:

Here are some examples for you to learn:
example of input and output of the system

Part 3. New planning problem:

Now you are given a new input planning problem as the following: The initial state and the goal of the problem in PDDL syntax. Choose a sequence of actions to accomplish this task. Only return the sequence of logical actions following the format of example output.

Example for a stacking problem

Prompt: Imagine you are a robot arm operator, you need to generate a sequence of actions to achieve the given goal. Here is the PDDL planning domain:

```
(define (domain LLM_generated_domain)
    (:requirements :strips :equality)
    (:predicates
        (box ?b1)
        (table ?t1)
        (above ?b1 ?b2)
        (holding ?b1 ?r1)
        (robot ?r1)
                                        ......
```

Here are some examples for you to learn:
Example input:
Initial state: (top, 2), (box, 2), (on_table, 2, 1), (top, 3), (box, 3), (on_table, 3, 1), (top, 4), (box, 4), (on_table, 4, 1), (table, 1), (not_holding, 0), (robot, 0)
Goal state: (top, 2), (box, 2), (above, 2, 3), (box, 3), (above, 3, 4), (box, 4), (on_table, 4, 1), (table, 1), (not_holding, 0), (robot, 0)
Example Output: pick(3, 1, 0), stack(3, 4, 0), pick(2, 1, 0), stack(2, 3, 0).

Now you are given a new planning problem as the following:
Initial state: (top, 2), (box, 2), (on_table, 2, 1), (top, 3), (box, 3), (on_table, 3, 1), (top, 4), (box, 4), (on_table, 4, 1), (table, 1), (not_holding, 0), (robot, 0)
Goal state: (top, 4), (box, 4), (above, 4, 2), (box, 2), (above, 2, 3), (box, 3), (on_table, 3, 1), (table, 1), (not_holding, 0), (robot, 0)
Choose a sequence of actions to accomplish this task, and return the action sequence following the example output provided.

LLM: pick(2, 1, 0), stack(2, 3, 0), pick(4, 1, 0), stack(4, 2, 0).

A.4. Experiment Tasks

A.4.1. Stacking

The stacking task involves collecting individual objects and placing them on top of each other to form stable stacks.

A.4.2. Unstacking

The unstacking task is the inverse process of stacking, requiring the robot to identify, grasp, and remove items from existing stacks without disturbing surrounding structures.

A.4.3. Rearrangement

The rearrangement task demands the robot to relocate objects from an initial configuration into a desired layout.

A.4.4. Alignment

The alignment task requires the robot to position multiple objects in a straight line with consistent spacing and orientation.

A.4.5. Color classification

In the color classification task, the robot must identify the color of each object, group them by color category, and stack or place them in designated areas accordingly.

A.4.6. Parts assembly

The parts assembly task involves recognizing components and sequentially assembling multiple machining parts together.

A.4.7. Tower of Hanoi

The Tower of Hanoi is a puzzle that involves moving a stack of disks from one base to another, one at a time, without ever placing a larger disk on top of a smaller one, using a third base as an auxiliary.

A.4.8. Bridge building

In bridge building, the robot is required to collect distributed blocks and configure it into a bridge structure.

A.4.9. Burger cooking

Lastly, for burger cooking, the robot needs to stack and pack the food ingredients together to make hamburgers.

A.5. Generalization Across Task Complexity

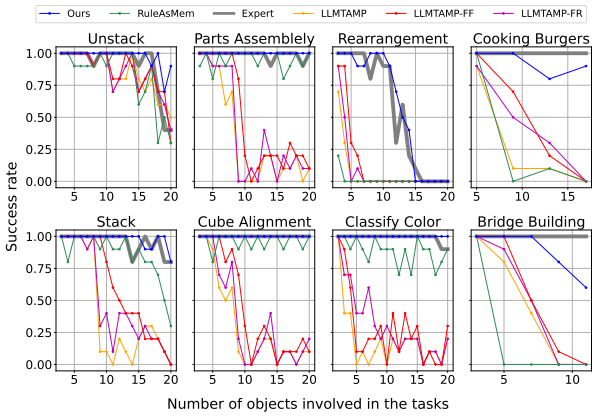

Figure 9. Planning success rate trend across increasing object counts for each task.

A.6. Time Cost of LLMTAMP Reasoning Model Variants

As noted in the paper, the reasoning models incur substantial computational overhead when generating task plans. To account for this, we extended the time limits for **o1-TAMP** and **R1-TAMP** in order to evaluate the contribution of reasoning to planning performance and to enable a fair comparison with our method. This section presents a comprehensive comparison of the time costs between the LLMTAMP variants and our approach.

As shown in Table 5, our method consistently yields lower average planning times across all tasks compared to the LLMTAMP reasoning model variants. This improvement is primarily attributed to PDDLLM's ability to structurally summarize the reasoning process into a standardized planning domain during a one-time offline inference step. As a result, no additional reasoning is required at test time. In contrast, LLMTAMP variants conduct reasoning independently for each task instance, leading to significantly higher computational costs. Such overhead makes these models impractical for deployment on physical robots, where real-time planning capabilities are often essential. As shown in Table 3, while **o1-TAMP** achieves comparable performance in certain tasks, the substantially higher time cost undermines its practical value, particularly in real robotic scenarios.

Table 5. Comparison of time cost between PDDLLM and LLM-TAMP reasoning variants

Experiment	Ours	o1-TAMP	o1-TAMP
Average Planning Time Cost (Second)			
Stack	5.94	39.22	211.40
Rearrangement	15.74	92.83	337.10
Tower of Hanoi	4.29	167.82	353.88
Bridge Building	6.45	82.47	305.07

A.7. Prompt Variation Test

In addition to the five main research questions discussed in the paper, we further evaluated the robustness of our method under variations in the prompting styles. Four different cases were tested to assess the stability of domain generation: parallel prompting with varying numbers of prompts, altering the prompting sequence of simulation outcomes, and tuning the prompting template.

A.7.1. Parallel prompting for predicate imagination

In this prompt variation, we perform parallel prompting of the LLM multiple times to obtain diverse responses simultaneously. These outputs are subsequently analyzed and aggregated to synthesize an optimal solution. While the core prompting procedure follows the same structure as described in Appendix A.3.1, additional post-processing steps are applied to summarize and consolidate the results. For prompt naming, we further prompt the LLM to select the most suitable name from among the parallel outputs. The prompt template is shown as the following:

Here is a list of predicate describing the same object state: predicate list. Please choose the PDDL predicate from the provided ones to best describe the scenario. Return the chosen one in PDDL syntax.

For relevance classification, if the parallel outputs are inconsistent, PDDLLM selects the majority response. In the case of a tie, it randomly selects one among the tied options.

Example for a choosing predicate from 5 parallel outputs

Prompt: Here is a list of predicate describing the same object state: [(above, ?o1, ?o2), (is_on, ?o1, ?o2), (aligned_vertically, ?o1, ?o2), (on_top_of, ?o1, ?o2), (above_object, ?o1, ?o2)]. Please choose the PDDL predicate from the provided ones to best describe the scenario. Return the chosen one in PDDL syntax.

LLM: (above, ?o1, ?o2)

Example for a choosing predicate from 10 parallel outputs

Prompt: Here is a list of predicate describing the same object state: [(above, ?o1, ?o2), (is_on, ?o1, ?o2), (aligned_vertically, ?o1, ?o2), (on_top_of, ?o1, ?o2), (above_object, ?o1, ?o2), (above, ?o1, ?o2), (on, ?o1, ?o2), (vertically_on, ?o1, ?o2), (on_top, ?o1, ?o2), (upper, ?o1, ?o2)]. Please choose the PDDL predicate from the provided ones to best describe the scenario. Return the chosen one in PDDL syntax.

LLM: (on, ?o1, ?o2)

A.7.2. Altering the prompting sequence of simulation outcomes

In this prompt variation, we alter the ordering of dimensions presented to the LLM. While the prompt template remains similar to that described in Appendix A.3.1, we do not adhere to a fixed dimension sequence. Instead, we shuffle the order of dimensions within the prompt to evaluate whether this affects the generated output. Some examples are provided here.

Example of prompting object position relation predicate with altered sequence - A

Prompt: There are two objects in the environment, whose positions are (x_1, y_1) and (x_2, y_2). In the horizontal direction, $|x_1 - x_2| \leq u$. In the vertical direction, $u < y_1 - y_2 \leq 2u$. Please create a predicate in PDDL syntax to describe this relation and classify if it is related to the task of stacking cubes together. Please return the result in the following format: predicate, relevance.
LLM: is_on($?o_1$, $?o_2$), related.

Example of prompting object position relation predicate with altered sequence - B

Prompt: There are two objects in the environment, whose positions are (x_1, y_1) and (x_2, y_2). In the vertical direction, $u < y_1 - y_2 \leq 2u$. In the horizontal direction, $|x_1 - x_2| \leq u$. Please create a predicate in PDDL syntax to describe this relation and classify if it is related to the task of stacking cubes together. Please return the result in the following format: predicate, relevance.
LLM: is_on($?o_1$, $?o_2$), related.

A.7.3. Template tuning

In this experiment, we aim to evaluate whether slight modifications to the prompt template affect the LLM's ability to generate predicate names and assess their relevance. Here is the fine-tuned template:

There are n objects in the environment, whose feature name are feature value 1 and feature value 2. In dimension 1, we know feature subspace range in dimension 1. In dimension 2, we know feature subspace range in dimension 2. Please create a predicate in PDDL syntax to describe this relation. Assign a score to this predicate indicating its relevance to the task of current task description. The score range is 0 to 1, where 0 indicate irrelevant and 1 indicate very relevant. Please return the result in the following format: predicate, score.

After collecting the scores, we set a threshold to determine which predicates are relevant. In the experiment, the threshold chosen is 0.5.

Example of prompting object position relation predicate with modified template

Prompt: There are two objects in the environment, whose positions are (x_1, y_1) and (x_2, y_2). In the horizontal direction, we know $|x_1 - x_2| \leq u$. In the vertical direction, we know $u < y_1 - y_2 \leq 2u$. Please create a predicate in PDDL syntax to describe this relation. Assign a score to this predicate indicating its relevance to the task of stacking cubes together. The score range is 0 to 1, where 0 indicate irrelevant and 1 indicate very relevant. Please return the result in the following format: predicate, score.
LLM: above($?o_1$, $?o_2$), 0.9.

The results, presented in Table 6, demonstrate that our method remains robust across different prompting styles. Regardless of the prompt variation, the generated planning domains consistently solve the test tasks with a success rate

approaching 100%. Minor fluctuations are attributed to randomness in the planning search process and occasional motion execution failures.

Table 6. Planning success rate for domains generated using different prompt styles.

Experiment	10-Parallel	5-Parallel	Sequence Altering	Template Tuning
Stack	97.8%	96.1%	96.1%	95.6%
Unstack	97.8%	100%	98.9%	98.3%
Cube Alignment	100%	100%	100%	100%

A.8. Planning Time Limit Variation

Table 7. Planning success rate (%) across tasks for all methods (Time limit = 25 s).

Method	Expert	LLMTAMP	LLMTAMP-FF	LLMTAMP-FR	RuleAsMem	PDDLLM
Stack	95.5 ± 0.6	41.5 ± 4.3	56.8 ± 3.4	43.8 ± 3.8	84.3 ± 3.3	97.5 ± 1.6
Unstack	87.3 ± 0.8	81.2 ± 1.1	85.0 ± 5.7	85.5 ± 5.4	79.8 ± 1.9	94.9 ± 0.5
Color Classification	96.3 ± 0.1	18.1 ± 1.5	24.9 ± 0.8	23.1 ± 3.3	87.6 ± 1.9	99.5 ± 0.4
Alignment	100.0 ± 0.0	31.6 ± 3.1	40.9 ± 2.0	35.3 ± 2.0	96.0 ± 0.8	100.0 ± 0.0
Parts Assembly	98.9 ± 0.6	33.6 ± 1.5	46.1 ± 2.1	37.5 ± 1.4	95.1 ± 0.6	100.0 ± 0.0
Rearrange	51.2 ± 2.7	5.6 ± 1.1	11.7 ± 0.6	7.4 ± 1.1	1.1 ± 0.6	52.5 ± 2.2
Burger Cooking	100.0 ± 0.0	27.8 ± 2.8	45.1 ± 7.3	38.9 ± 5.6	27.8 ± 2.8	89.6 ± 3.2
Bridge Building	100.0 ± 0.0	43.3 ± 3.3	48.9 ± 5.9	47.2 ± 4.3	20.0 ± 0.0	87.2 ± 4.3
Tower of Hanoi	83.3 ± 2.4	14.3 ± 0.0	14.3 ± 0.0	14.3 ± 0.0	14.3 ± 0.0	85.7 ± 0.0
Overall	89.0 ± 0.3	34.5 ± 0.4	43.2 ± 0.8	38.1 ± 0.9	68.3 ± 0.9	90.5 ± 0.9

Table 8. Planning success rate (%) across tasks for all methods (Time limit = 100 s).

Method	Expert	LLMTAMP	LLMTAMP-FF	LLMTAMP-FR	RuleAsMem	PDDLLM
Stack	97.7 ± 0.7	41.5 ± 4.3	76.7 ± 2.7	71.0 ± 2.7	85.5 ± 2.9	97.5 ± 1.6
Unstack	96.1 ± 0.2	90.3 ± 1.5	96.9 ± 1.2	96.1 ± 1.1	88.4 ± 1.2	97.7 ± 0.7
Color Classification	100.0 ± 0.0	18.1 ± 1.5	42.0 ± 1.9	64.0 ± 2.2	88.7 ± 2.3	100.0 ± 0.0
Alignment	100.0 ± 0.0	31.6 ± 3.1	55.7 ± 3.8	44.3 ± 3.8	96.0 ± 0.8	100.0 ± 0.0
Parts Assembly	98.9 ± 0.6	33.6 ± 1.5	57.1 ± 0.8	47.7 ± 2.5	95.1 ± 0.6	100.0 ± 0.0
Rearrange	69.3 ± 1.3	5.6 ± 1.1	17.4 ± 1.1	14.7 ± 1.2	1.1 ± 0.6	69.4 ± 0.5
Burger Cooking	100.0 ± 0.0	27.8 ± 2.8	50.0 ± 4.8	51.4 ± 6.1	27.8 ± 2.8	97.2 ± 2.8
Bridge Building	100.0 ± 0.0	43.3 ± 3.3	53.3 ± 3.8	57.8 ± 2.2	20.0 ± 0.0	87.2 ± 4.3
Tower of Hanoi	100.0 ± 0.0	14.3 ± 0.0	14.3 ± 0.0	14.3 ± 0.0	14.3 ± 0.0	100.0 ± 0.0
Overall	94.4 ± 0.2	35.8 ± 0.4	55.6 ± 1.2	54.8 ± 0.7	69.9 ± 0.7	94.2 ± 0.5

As outlined in the experimental design section, we set a default planning time limit of 50 seconds to reflect the real-time constraints commonly imposed on robotic systems. Although prior studies typically allow planning times on the order of minutes [16, 19, 31], the exact limits vary considerably. To assess the robustness of our approach under different time allowances, we evaluate performance at time limits of 25, 50, and 100 seconds. The results show that our method consistently outperforms all LLM-based baselines across all tested time settings and task types. Moreover, the planning domains derived by PDDLLM yield performance comparable to that of expert-designed domains under all time settings and across all tasks. All simulations were conducted on a system with an Intel Core i9-14900KF CPU without GPU acceleration, and all LLM prompting was performed using API services.

A.9. Example of PDDLLM imagined predicates

Table 9. Examples of imagined predicates across multiple categories.

Predicate Name

Object Position Predicates

```
(above ?a ?b)
(forall ?a (above ?a ?b))
(forall ?a (above ?a ?b) is false)
(not (above ?a ?b))
(beside ?a ?b)
(forall ?a (beside ?a ?b))
(forall ?a (beside ?a ?b) is false)
(not (beside ?a ?b))
```

Object Support Predicates

```
(on-table ?a ?t)
(not (on-table ?a ?t))
(forall ?a (on-table ?a ?t))
(forall ?a (on-table ?a ?t) is false)
(aligned-x ?a ?t)
(not (aligned-x ?a ?t))
(forall ?a (aligned-x ?a ?t))
(forall ?a (aligned-x ?a ?t) is false)
```

Robot Predicates

```
(holding ?a ?r)
(not (holding ?a ?r))
(forall ?a (holding ?a ?r))
(forall ?a (holding ?a ?r) is false)
(gripper-near-open ?a ?r)
(not (gripper-near-open ?a ?r))
(forall ?a (gripper-near-open ?a ?r))
(forall ?a (gripper-near-open ?a ?r) is false)
```

Color Predicates

```
similar-color ?a ?b
(not similar-color ?a ?b)
(forall ?a similar-color ?a ?b)
(forall ?a similar-color ?a ?b is false)
distinct-colors ?a ?b
(not distinct-colors ?a ?b)
(forall ?a distinct-colors ?a ?b)
(forall ?a distinct-colors ?a ?b is false)
```

Size Predicates

```
(smaller ?a ?b)
(not (smaller ?a ?b))
(forall ?a (smaller ?a ?b))
(forall ?a (smaller ?a ?b) is false)
(larger ?a ?b)
(not (larger ?a ?b))
(forall ?a (larger ?a ?b))
(forall ?a (larger ?a ?b) is false)
```