Verbalized Machine Learning with Best-of-N Sampling Generates Better World Models for Task Planning

Zhouliang Yu[1,*] Yuhuan Yuan[2,*] Tim Z. Xiao[3] Fuxiang Frank Xia[4] Jie Fu[5]
Ge Zhang[6] Ge Lin[2] Weiyang Liu[3]

[1]The Chinese University of Hong Kong
[2]The Hong Kong University of Science and Technology(Guangzhou)
[3]Max Planck Institute for Intelligent Systems, Tübingen
[4]Economic and Social Research Institute [5]Shanghai Artificial Intelligence Laboratory [6]M-A-P

Abstract

Solving complex planning problems requires Large Language Models (LLMs) to explicitly model the state transition to avoid rule violations, comply with constraints, and ensure optimality—a task hindered by the inherent ambiguity of natural language. To overcome such ambiguity, Planning Domain Definition Language (PDDL) is leveraged as a planning abstraction that enables precise and formal state descriptions. With PDDL, we can generate a symbolic world model where classic searching algorithms, such as A$*$, can be seamlessly used to find optimal plans. However, directly generating PDDL domains with current LLMs remains an open challenge due to the lack of PDDL training data. To address this challenge, we propose to scale up the test-time computation of LLMs to enhance their PDDL reasoning capabilities, thereby enabling the generation of high-quality PDDL domains. Specifically, we introduce a simple yet effective algorithm, which first employs a Best-of-N sampling approach to improve quality of the initial solution and then refines the solution in a fine-grained manner with verbalized machine learning. Our method outperforms o1-mini by a considerable margin in PDDL domain generation, achieving over 50% success rate on two tasks (*i.e.*, generating PDDL domains from natural language description or PDDL problems). This is done without requiring additional training. By taking advantage of PDDL as state abstraction, our method is able to outperform current state-of-the-art methods on almost all the competition-level planning tasks.

1 Introduction

Enabling large language models (LLMs) to plan in complex scenarios like Barman, Floortile, and Termes remains an open problem. While recent LLMs like OpenAI-o1 excel at complex reasoning tasks, including coding and mathematics, they still struggle with deductive reasoning and principled planning that requires the consideration of optimality, constraints, and complex state transitions. Our experiments with o1 demonstrate this limitation persists even when enhanced with self-critique techniques and multiple re-sampling. A natural solution is to translate the world abstraction from natural language to Planning Domain Definition Language (PDDL), which uses first-order logic (FOL) to explicitly describe states and relationships. Compared to natural language, the formal language nature of PDDL makes verification easy, and also enables the precise specification of constraints and objectives for seamless integration of off-the-shelf planning algorithms. However, it remains a huge challenge to translate natural language descriptions into PDDL domains in a satisfactory

[*]Equal contribution. Project page: xxx

accuracy. Current LLMs perform poorly in this translation task due to two key challenges: the scarcity of high-quality PDDL training data and the complexity of maintaining logical consistency across predicates and actions. Traditionally, the translation process has heavily relied on human expertise and manual refinement, making it difficult to automate and scale [GVSK23].

To address these challenges, our work leverages LLMs to generate PDDL-based symbolic world models for task planning without requiring model finetuning. We achieve this through a simple yet effective strategy to scale the test-time computation of LLMs. Specifically, we start by generating multiple PDDL domains from the query using Best-of-N (BoN) sampling. This step aims to have a good initial solution that is error-free and logically correct. Then we refine the best initial solution using verbalized machine learning (VML) [XBSL24] such that the generated PDDL domain can best fit the input query.

Our method is guided by the insight that a good trade-off between VML is a test-time training approach that involves a generator LLM that produces PDDL domains, and critic LLMs that provide feedback on logic and grammar errors in the current iterations of solutions. However, the performance of VML is highly dependent on solution initialization; poor initialization can lead to slow convergence or suboptimal solutions. Best-of-N (BoN) sampling emphasizes exploration over exploitation since it does not take into account past predictions from the LLM generator. In contrast, single-beam VML is an exploitation-only algorithm that does not consider new solution spaces. To balance exploration and exploitation, we use BoN to generate multiple diverse candidates as initial solutions for VML. This hybrid approach allows us to improve overall performance by combining the diverse solution generation of BoN with the iterative refinement capabilities of VML. Our major contributions are listed below:

- **Scalable PDDL Domain Generation**: We leverage VML test-time scaling methods that significantly outperform existing methods without additional training. Our approach achieves state-of-the-art performance with 97.5% success rate on NL2Domain task and 92.7% on Prob2Domain by using Qwen2.5-Coder-7B as the base model, substantially surpassing o1's performance (41.7% and 33.7% respectively).
- **Superior Planning through PDDL Abstraction**: We demonstrate that PDDL-based formal abstraction enables more robust planning compared to direct LLM-based planning. Our method successfully handles complex domains like Barman and Termes, where o1-as-planner approaches fail.
- **Efficient Test Time Compute Scaling**: We equip the VML with BoN sampling as initialization, which efficiently balances exploration and exploitation, leading to faster convergence to more optimal solutions.

2 Related Works

2.1 Limitations of LLMs on Planning

Recently, LLMs have been widely used for planning. However, The results on Planbench [VMO+24, VSK24], and other constraint-heavy and spatially complex environments [WLB+24] show that LLMs still cannot do principled reasoning and planning. Principled planning demands a sophisticated understanding of logical and theoretical constructs to tackle complex problems. A recent study [Mir24] transformed the math word problem into symbolic representations and found LLMs have a significant drop when reasoning the same problem in symbolic representations. LLMs are trying to construct completions to prompts by probabilistically predicting the next word based on sequences learned from a vast dataset, rather than engaging in logical deduction or structured inferencing [Kam24].

The research community come up with several approaches to help LLMs plan. Firstly, supervised fine-tuning of the LLMs by the casual reasoning traces *e.g.*chain-of-thought style data [YXWK24]. This approach reinforces LLMs COT generation abilities, but within each step still contains hallucinations. Another series of approaches is LLMs-as-verifier, which either fine-tunes an LLM to be a process reward model [ZHB+24] to self-improve or uses LLMs as a sparse objective reward model to judge the final results [ZCS+23]. This approach can effectively check the inherent common sense and knowledge errors if the verifier has been trained with the corresponding corpus, however, is limited to being transferred to deductive reasoning. LLMs cannot trustfully tell if the intermediate step is wrong only when they can explicitly model its consequence. [HGM+23] uses LLMs as a world model (WM) that estimates the state transitions when planning. However, the LLM-based WM mimics reasoning by generating plausible outputs based on observed patterns that are not explicitly aligned with the evolving of the physical world. Given the challenges mentioned above, the primary research problem we aim to investigate is: *Is there an approach to generate explicit WMs at scale automatically by LLMs?* To explicitly model the world we adopt the Planning Domain Definition Language (PDDL) [McD00] as an intermediate abstraction to construct the WM.

Every element in PDDL *e.g.*actions, predicates, initial states, and goals are described using first-order logic(FOL). The advantages of PDDL are: 1. Easy validation: FoL enables automated and precise logical validity, PDDL parsers ensure that the FOL expressions used are well-formed and adhere to the expected logical structure. Also, planners can use validation tools to verify that the series of actions indeed leads from the initial to the goal state according to the specified logic. 2. Precise state transition expression: PDDL uses FOL to define the state of the world, enabling a structured representation of conditions that must hold true for actions and states. This helps in modeling the world in a detailed and unambiguous manner. However, the barrier is that current SOTA LLMs cannot generate PDDL codes with high accuracy. We demonstrate that by leveraging test-time scaling, specifically through verbalized machine learning with BoN sampling to search for more optimal initializations, LLMs can achieve human-expert-level performance in synthesizing IPC-level PDDL domains. In the end, with the generated PDDL domains, we can leverage external planners *e.g.*A$*$ to search for optimal solutions.

2.2 World Model Generation

We are not the first to leverage LLMs for WM generation. GIF-MCTS [DMAM24] adopts LLMs to translate natural languages that describe details of the actions, observations, and rewards within the environment to Gym-like [BCP+16] Python code. They use pre-collected trajectories as online testing samples to evaluate the effectiveness of the Python code. World-coder [TKE24] reduces the number of samples by introducing optimistic learning objectives when facing uncertainty. The generated WM can be unreliable. World-coders use self-refinement through human-crafted prompt engineering to fix errors. Before the emergence of LLMs, [BG20] takes directed graphs that encode the state space structure of certain problem instances as inputs, and leverages a two-level combinatorial search to infer the PDDL domains. The most similar works to others are [GVSK23] and [ZVL+24]. We list the comparison between our methods with the above two works in Table 1:

Our work distinguishes itself from previous studies by focusing on the generation of complete PDDL domains. In contrast, [GVSK23] concentrates on generating each action individually, and Planetarium emphasizes the creation of PDDL problem instances. From a technical perspective, [GVSK23] incorporates multiple human experts in the loop to refine the generated actions, and Planetarium leverages an extensive amount of training data to achieve its objectives. In contrast, our approach prioritizes automation and scalability across diverse planning scenarios without extensive training. We employ a training-free, slow-thinking methodology that harnesses the power of generating

Table 1: Comparison of PDDL Synthesis Methods

Methods	Synthesis Objective	Benchmarking	Technical Methods
[GVSK23]	Each action separately within PDDL domains	3 domains: Household, Logistics, Tyreworld	Several graduate students to guide
[ZVL+24]	Whole PDDL problems (initial state and goal)	2 domains: Gripper, Blockworld with thousands of problems	SFT on a dataset of over 132,037 instances
Ours	Whole PDDL domains	283 IPC domains (NL2Domain), 332 IPC domains (Prob2Domain)	Test time scaling without the need for a large training set and human expert-in-the-loop

accurate PDDL codes.

3 Method

We now describe the methodology of our work. Figure 3.1 is an overview, the detail problem formulation can be found at Appendix B

3.1 PDDL as an Abstraction

We use PDDL (Planning Domain Definition Language) as an intermediate abstraction for planning purposes. PDDL uses first-order logic to represent comprehensive states, actions, and transitions in a structured manner The states are expressed through a set of predicates that describe the properties of objects in the environment. Each predicate represents a relationship or characteristic, such as `on(A, B)`, which indicates that object A is on top of object B. A state can be represented as:

$$S_0 = \{\text{on}(A, B), \text{clear}(C)\}$$

This representation includes relationships that capture the positions and statuses of the objects within the environment. Actions in PDDL are tightly bound to the representation of states through the use of preconditions and effects. Each action is defined by specifying what must be true in the current state for the action to be applicable (preconditions), as well as what changes in the state when the action is performed (effects). For example, consider an action `move(A, B, C)`, indicating that move the object A from B to C. The preconditions and effects could be defined as follows: Preconditions: $\text{Pre}(\text{move}(A, B, C)) = \{\text{on}(A, B), \text{clear}(C)\}$and Effects: $\text{Eff}(\text{move}(A, B, C)) = \{\text{on}(A, C), \text{clear}(B)\}$. They indicate that for the action `move(A, B, C)` to be executed, object A must be positioned on B, and C must be clear of any objects. And after moving A from B to C, A is now on C, and B is clear. The transition can be formulated as: $T(S, A) \rightarrow S'$. State transitions occur according to a defined sequence of actions that an agent may take, leading to new configurations of the world. For instance if the action $\text{move}(A, B, C)$ on S_0, the transition can be represented as:

$$S_1 = T(S_0, \text{move}(A, B, C)) \rightarrow \{\text{on}(A, C), \text{clear}(B)\}$$

By repeatedly exploring all possible actions, we can form a state transition graph consisting of nodes (states) and directed edges (actions) that connect these nodes based on the actions that lead to different states. We continue this process by expanding the graph from the newly added nodes until the goal state is reached and added to the graph, or there are no more actions left to explore. we apply classical algorithms A$*$ Search to plan efficiently.

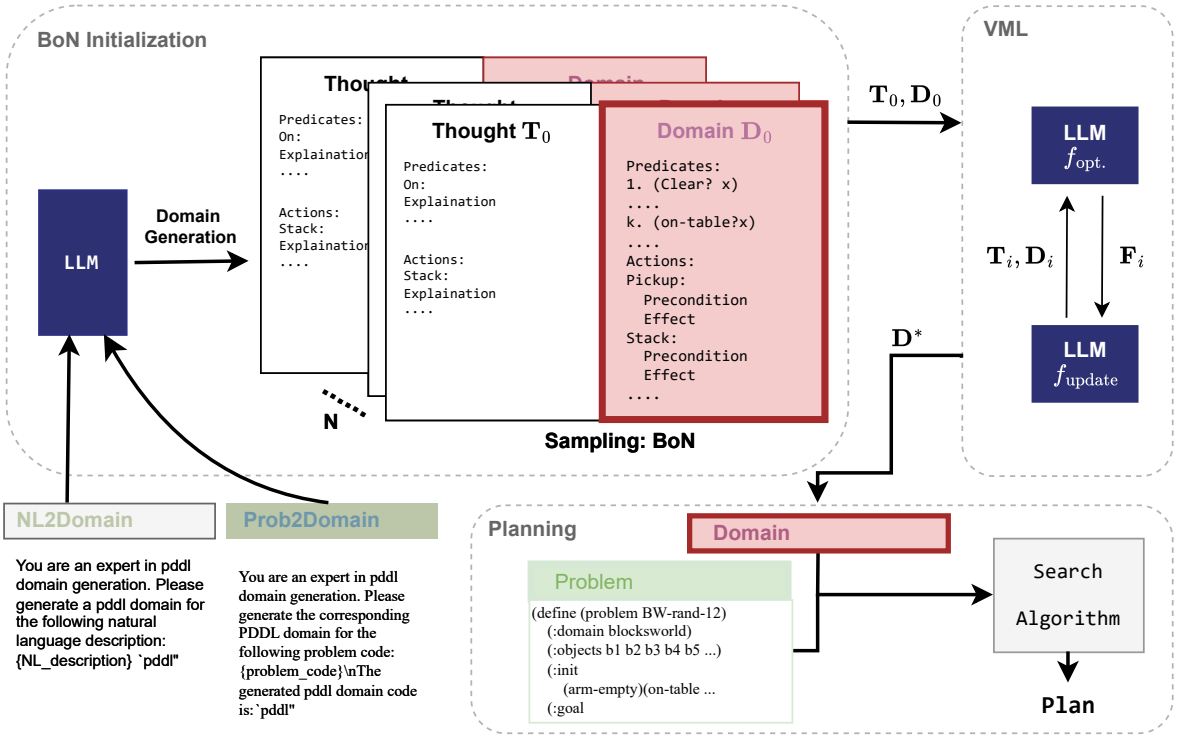

Figure 3.1: The overall pipeline of our methods. We adopt formal language-PDDL as an intermediate abstraction for planning. The upper domain generation process shows how we generate an error-free and logically correct PDDL domain, which involves reducing the search space by focusing LLMs on predicates and actions. We employ BoN sampling to consider multiple possible domain actions simultaneously. We then use verbalized machine learning to iteratively refine and improve the sampled domains in the text space. The downward planning blocks show how we generate the plan. Once we have a synthetic PDDL domain that passes PDDL grammar checks, we combine it with a PDDL problem, then leverage searching algorithms such as A$*$ to search for the optimal plan.

3.2 Explanation Chain of Thought

The explanation chain of thought approach involves a simple three-step process: 1. `Focusing`: By narrowing the model's attention to the most pertinent variables, the model can more efficiently determine which elements can function as predicates and actions. This focused approach helps manage the problem space's complexity and dimensionality, potentially leading to more precise and effective outcomes. 2. `Explaination`: Provide a transparent and explainable rationale for the model's decisions on predicates and actions, and mitigate the risk of loss of vital contextual information during the generation of the final PDDL code. 3. `Autoformalization`: LLMs autoformalize the predicates and actions to form a complete PDDL domain code. Similar methods of reducing the search space dimensionality and explaining the model decision have been implemented in System 2 Attention [WS23] and Plan-searh [Z+24]. The explanation-COT provides a comprehensive context of planning domain generations.

3.3 Test-time Scaling by Best-of-N Sampling and Verbalized Machine learning

Our proposed test-time scaling approach is a verbalized machine learning [XBSL24] method with best-of-N sampling for better initialization points.

Best-of-N sampling. For each problem, the LLMs generate N candidate solutions in parallel and retain the K samples with the highest log-likelihoods. This process involves three main steps: candidate generation, scoring, and selection. A high temperature is introduced to add more randomness and diversity during sampling. Each candidate c_i is assigned a score S_i based on the sum of the log-likelihoods of its generated tokens:

$$S_i = \sum_{t=1}^{L_i} \log p_t \left(w_t^{(i)} \right), \quad \forall i \in \{1, 2, \ldots, N\}$$

where:L_i is the length of candidate c_i, $w_t^{(i)}$ is the t-th token of candidate c_i, $p_t \left(w_t^{(i)} \right)$ is the probability of token $w_t^{(i)}$ at position t. During the selection phase, the top K candidates with the highest scores are chosen and sent to the syntax checker to further select the grammar error-free PDDL domains, before being deployed for planning.

Verbalized Machine Learning. With the BoN sampling to select the candidates as the initialize solution, we use verbalized machine learning (VML; [XBSL24]) to refine both the generated PDDL domain and the explanation chain of thought introduced in Section 3.2. In VML, functions are parameterized using natural language rather than numerical values. Using an LLM as the inference engine, we can evaluate such a natural language parameterized function, and optimize it with respect to its parameters in the natural language space. In our setting, we are given a description $\mathcal{G}$ from the planning domain either in natural language for NL2Domain or in problem code for Prob2Domain, and we want to generate the most accurate corresponding PDDL domain description $\mathbf{D}^*$, *i.e.*,

$$\mathbf{D}^* = \arg\min_{\mathbf{D}} \ \mathcal{L}(\mathcal{G}, \mathbf{D}) \tag{3.1}$$

where $\mathcal{L}(\cdot)$ is a loss function defining the closeness between $\mathcal{G}$ and $\mathbf{D}$. Solving Equation (3.1) is difficult as both $\mathcal{G}$ and $\mathbf{D}$ are text, and $\mathcal{L}(\cdot)$ is hard to define unless abstractly using natural language. Using the VML framework, we can approximate Equation (3.1) with an iterative algorithm that alternates between two natural language parameterized functions at iteration i:

$$\mathbf{F}_i = f_{\text{opt.}}(\mathcal{L}, \mathcal{G}, \mathbf{T}_{i-1}, \mathbf{D}_{i-1}), \tag{3.2}$$

$$\mathbf{T}_i, \mathbf{D}_i = f_{\text{update}}(\mathbf{F}_i, \mathbf{T}_{i-1}, \mathbf{D}_{i-1}), \tag{3.3}$$

where $\mathbf{T}_{i-1}$ and $\mathbf{D}_{i-1}$ correspond to the current thoughts and the current PDDL domain, $\mathbf{F}_i$ is the feedback from the optimizer function $f_{\text{opt.}}(\cdot)$, $\mathbf{T}_i$ and $\mathbf{D}_i$ are the updated thoughts and PDDL domain output from the update function $f_{\text{update}}(\cdot)$. $\mathbf{D}_0$ is initialized from the best-of-N sampleing. These two functions are evaluated through separate LLMs calls. The following shows an example prompt template for $f_{\text{opt.}}(\cdot)$.

Prompt template for $f_{\text{opt.}}(\cdot)$

You will be provided a natural language description of a planning domain, and its corresponding PDDL domain code with intermediate thoughts explaining each predicate and action. Your task is to generate critical feedback on the PDDL domain code based on the natural language description. You should evaluate the grammar and logic of the PDDL domain codes, and the logic error in the intermediate thoughts.
PDDL synthesis problem: $\{\mathcal{G}\}$
Back translation chain of thoughts: $\{\mathbf{T}_{i-1}\}$
Generated PDDL Domain: $\{\mathbf{D}_{i-1}\}$

VML with BoN initialization. BoN emphasizes exploration rather than exploitation. It maintains multiple diverse candidate solutions (beams) simultaneously to consider various regions of the search space. The disadvantage of BoN is: it shows early saturation to a lower performance than VML, as it samples the solution in randomly, and lacks trial and error procedure to learning from the failure cases. Single-beam VML without BoN as an initialization method focuses on exploiting the initial solution. It updates the initial solution iteratively in the direction of the gradient, effectively moving toward the gradient descent direction in the solution landscape. However, this local search strategy can easily become trapped in local minima, as it relies solely on local gradient information and does not inherently explore other regions of the search space. This explains why VML usually needs longer runs to find a good solution. To equip VML with exploration abilities, we introduce sampling parameters that enable diverse sampling: a high-temperature parameter, for both the generator LLM and the critic LLM. This allows the model to explore new solution spaces beyond the local neighborhood of the initial solution. However, the performance of VML is highly influenced by the initial solution; a poor initial solution may take more runs to converge. To address this issue, we use the solution generated by BoN as the initial solution, ensuring the quality of the starting point. VML is then used to self-improve and refine the solution. Our approach balances exploration and exploitation, enabling VML to escape local minima more effectively. Additionally, increasing the number of iterations allows VML to refine its solutions further and increases the opportunity to discover better optima.

4 Experiments

We conducted extensive experiments to compare our proposed test-time scaling methods with existing state-of-the-art LLMs on competition-level Planning Domain Definition Language (PDDL) domain synthesis tasks. Our methods consistently improve PDDL generation across almost every LLM considered, encompassing various model scales and types. By adopting PDDL as an abstraction layer, we transform the role of LLMs from planners to PDDL domain generators. With this synthetic abstraction, incorporating a traditional planner in the loop allows us to mitigate the hallucinations that can occur when using LLMs directly as planners.

4.1 Experiment Setup

Tasks and Datasets. We evaluate the test-time scaling methods on the International Planning Competition benchmark[1], which contains various complex planning domains and problems. Our evaluation focuses on two PDDL domain synthesis tasks: 1. NL2Domain: Generating a PDDL domain based on a natural language description. 2. Problem2Domain: Generating the necessary PDDL domain based on PDDL problems. The natural language descriptions for the PDDL domains are generated by GPT-4o.

Large Language Models. The baseline LLMs include: 1. Qwen2.5-instruct [YYH+24], which ranges from 0.5 billion to 72 billion parameters. 2. LLaMA3.1-Instruct [DJP+24], which consists of 8 billion and 70 billion parameter models. 3. Yi-1.5-chat [YCL+24], which includes models with 6 billion, 9 billion, and 34 billion parameters. We also consider code-oriented LLMs as baselines, including Qwen2.5-Coder and Yi-1.5-Coder. In addition to open-source LLMs, we compare against closed-source models from OpenAI, such as GPT-4o, o1-mini, and o1-preview. We apply test-time scaling to the Qwen series models and conduct all experiments in a zero-shot setting, without any further fine-tuning of the LLMs.

[1] https://github.com/potassco/pddl-instances

Chain of Thought Prompting. For our baselines, we utilize traditional chain-of-thought (COT) prompting. The current LLMs have been trained using the System 2 distillation approach [WS23]. During Supervised Fine-Tuning (SFT), these models are exposed to datasets that include step-by-step reasoning instead of just providing the final answers. This approach has the advantage of enabling the trained LLMs to generate detailed reasoning traces during inference automatically. In our methods, we employ explanation-based COT prompting first to identify the essential candidates for predicates and actions. Subsequently, we ask the LLM to explain the rationale behind its choices. The detailed prompts can be seen in Appendix G.

Sampling parameters. To obtain diverse PDDL domain synthesis paths, we applied temperature sampling with temperature $T = 0.7$ for both BoN and VML algorithm.

4.2 Main Results

The results are shown in Table 2. The BoN algorithm significantly enhances PDDL domain synthesis across almost all LLMs in the Qwen series, which range from 1.5B to 72B parameters. Interestingly, the performance gains are more pronounced in larger models. For instance, we observe a +9.5 improvement in the VAL pass rate for the NL2Domain task and a +0.3 improvement for the Prob2Domain task in the Qwen2.5-3B-Instruct model. In contrast, the Qwen2.5-32B-Instruct model shows remarkable improvements of +46.7 for NL2Domain and +39.5 for Prob2Domain.

However, We observed that increasing model size does not absolutely guarantee better performance in our tasks; for example, the Llama3.1-Instruct-70B only gets 0.0 on NL2Domain and 0.6 on Prob2Domain; the BoN-8-Qwen2.5-Instruct-72B model exhibits a −6.0 change in the NL2Domain compared to the BoN-8-Qwen2.5-Instruct-32B method. This suggests that while larger models can offer advantages, other factors *e.g.*training dataset may also influence outcomes. Notably, both the code LLMs have achieved higher success rates than the instruct model at equivalent scales, indicating that factors beyond sheer model size contribute to effectiveness.

Adopting VML beyond the BoN sampling enables a robust further improvement. Surprisingly, even for the largest model Qwen2.5-Instruct-72B under BoN@8 can still be further improved by VML, *i.e.*+17.6 on NL2Domain and +12.6 on Prob2Domain.

4.3 Compare BoN versus VML

This section compares the convergence performance of BoN with VML. And showcases to illustrate why VML outperforms BoN in our tasks. The computational budgets are primarily influenced by two key factors: N: The number of searches conducted via BoN sampling. E: The number of epochs utilized in VML. The objective of this study is to analyze how variations in both variables affect the performance of synthesizing PDDL, and in what circumstances, they would converge. We leverage the Qwen2.5-7B-Coder as the base model. The results are shown in Figure 4.1.

Performance generally improves as N increases from 1 to 32. However, as N continues to grow beyond this point, reaching 256, the performance levels off. This suggests an optimal range for N where the performance benefits are maximized without incurring unnecessary computational overhead. For VML, as the number of epochs increases, both NL2Domain and Prob2Domain show an upward trend, suggesting that the self-improvement process is effective in enhancing the performance of domain generation, although the rate of increase remains relatively steady. Finally, the percentage of accurate domains converges at 97.5% for NL2Domain and 92.7% for Prob2Domain. BoN shows earlier saturations than VML.

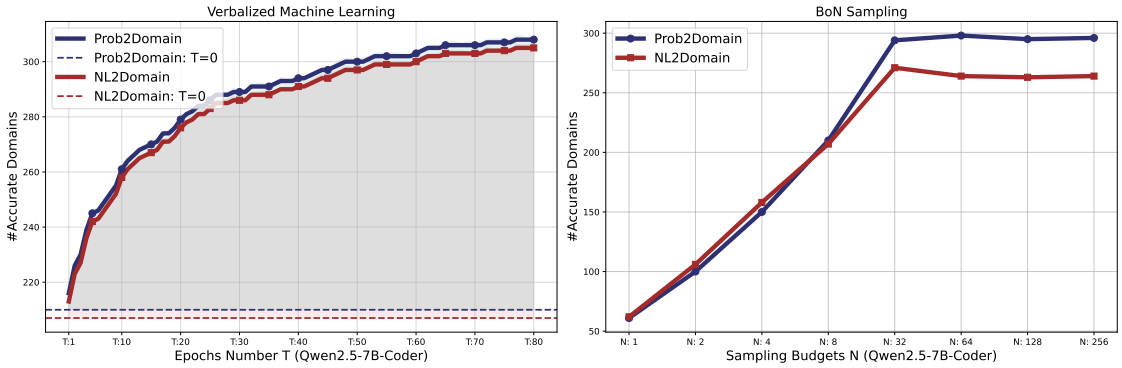

Figure 4.1: Left: Performance trend with increasing update epochs. Right: The trend of increasing sampling numbers.

The advantage of VML over BoN can be attributed to its nature as a multi-iteration in-context learning mechanism. Unlike BoN, which may suffer from logic errors due to sampling once per candidate without feedback, VML provides a comprehensive context enriched with self-critique feedback. In the specific domain of PDDL synthesis—which bridges code translation (autoformalization) and reasoning—VML's capabilities are particularly advantageous. This task surpasses simple one-to-one translations or non-common sense symbolic deductive reasoning, requiring an understanding of both formal and natural language nuances. Generating criticism for PDDL can be challenging, especially when dealing with formal languages that lack explanatory depth. However, the chain-of-thought feature in our method resolves this issue by offering detailed natural language explanations for each step of the reasoning process. This not only facilitates better understanding but also enables iterative refinement of both the model's natural language output (explanation-cot) and formal language output (PDDL domain), thereby reducing logical errors and improving the quality of domain synthesis. A case study comparing BoN and VML, illustrated in Table 3, further supports this conclusion.

4.4 Ablate the Initialization Settings

This section ablates the VML under various LLM optimizers and solution initialization settings. We test three LLMs: 1. Qwen2.5-Coder-7B, 2. Deepseek-Coder-7B-Instruct-v1.5 [GZY+24], 3. LLaMa-3.1-8B-Instruct. We observe the following pattern: 1. VML demonstrates significant sensitivity to the initialization method. Specifically, using direct sampling for solution generation leads to suboptimal outcomes, with fewer accurate PDDL domains synthesized compared to when the BoN method is used for initialization. 2. Employing BoN as an initialization technique facilitates faster convergence in the VML process, thereby enhancing its effectiveness. 3. VML does not improve the performance of LLaMa or the BoN method itself. LLaMA exhibits a zero success rate in the single-pass setting, leading us to hypothesize that LLaMA may lack the necessary foundational knowledge or task-specific understanding required for effective performance in this context. This deficiency could stem from insufficient training on relevant data or an inadequate architecture for handling the complexity of the tasks at hand. This suggests that for complex reasoning tasks, even with the benefits provided by VML, LLMs require a foundational understanding and common sense knowledge pertinent to the task at hand.

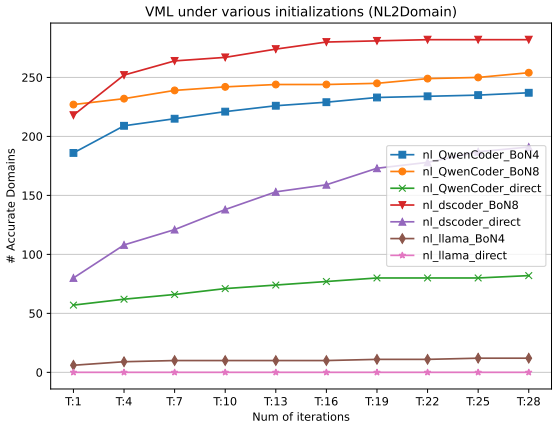

Figure 4.2: The trend of VML on NL2Domain tasks with various initialization settings.

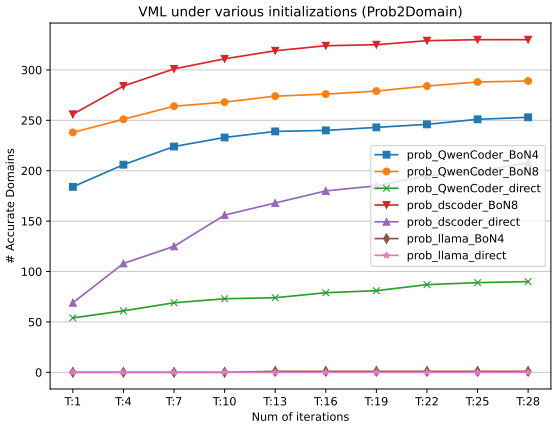

Figure 4.3: The trend of VML on Prob2Domain tasks with various initialization settings.

4.5 PDDL Problem Generations

Our method can effectively generalize to PDDL problem generation. We adopt the benchmark from Planetarium [ZVL+24], which focuses on generating PDDL problems from natural language descriptions. The dataset is complex due to its abstractness and the longer context required for describing objects. The baseline methods include GPT-4, Gemma 1.1 IT models [TMH+24] with 2B and 7B parameters, and Mistral [JSM+23] v0.3 Instruct 7B, both evaluated in zero-shot and fine-tuned settings. The results of baselines in Table 4 are originally from Planetarium. Notably, Planetarium fine-tuned Gemma and Mistral on a training dataset of 132,027 examples of PDDL problem code dataset, which raises concerns about overfitting, as the correct rate increases sharply from close to 0.0% to over 94%. Our methods based on Qwen2.5-Coder-7B attain a correct rate of 99.24% by BoN@16, and 99.60% by VML@1 with BoN@16 as the solution initialization method, illustrating scaling up test-time computation can efficiently achieve more optimal performance on PDDL problem generation.

5 Comparing Our Methods against LLM-as-planner Methods

In the previous sections, we demonstrated that our methods can generate PDDL domains that pass the grammar check tool, VAL, even for highly complex domains *i.e.*IPC-level. In this section, we present the advantage of applying synthesized PDDL domains for mode-based planning rather than the method that uses LLMs directly as planners. For a comprehensive description of all planning tasks in natural language, please refer to Appendix C.

5.1 LLMs are Limited at Planning in Complex Senerios

o1 frequently violates rules when facing complex planning problems *e.g.*Termes, Barman, etc. To investigate if this issue stems from prompt engineering, we provided the LLMs with clear instructions on the rules they must adhere to and instructed them to check rule compliance at each planning step. We aimed to prevent errors caused by either the model's misunderstanding of specific rules or users' failure to request strict adherence to those rules.

Identifying where LLMs err is easier than ensuring error-free planning. Therefore, we introduced an additional self-critique phase, allowing the LLMs to review and refine their plans based on their

assessments. We also implemented multiple sampling—up to eight—to see if rejection sampling could generate accurate plans.

We observed the following patterns when the LLM plans: 1. Despite explicitly informing LLMs to avoid breaking any rules, the generated plans still contain elements that inherently violate the rules. 2. When self-critique, LLMs claim that the plan is valid and optimal, even when not. 3. Sampling multiple solutions raises the success rate, however, still causes frequent hallucinations. The example case of o1 failing to solve Termes is shown in Figure 5.1.

5.2 Our Methods on Planning

o1-as-planner uses natural language as an abstraction and directly generates plans via sampling in its internal knowledge space. In contrast, our approach utilizes symbolic formal language as an abstraction. In our approach, we begin by translating the natural language input, which describes the planning domain, into PDDL representation, using the verbalized machine learning method. This abstraction then serves as the basis for model-based planning, where we employ search algorithms to identify an optimal plan. An example of the explicit state-based search graph used during planning for Termes can be found in Appendix F. The numerical comparison results against the LLM-as-planner method are in Table 5.

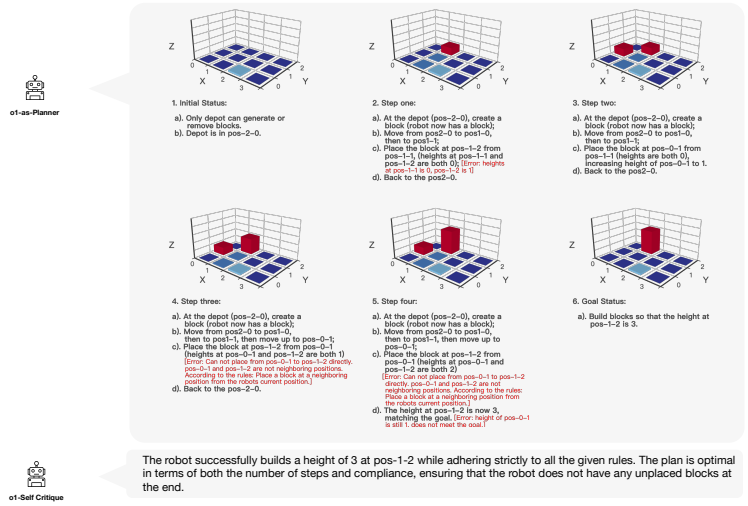

Figure 5.1: o1-as-planner on Termes: o1 frequently hallucinate during planning. For example at step one, o1 moves an "imagined" block to pos 1-2. A similar situation happened in step three and step four. However, when self-critique, the critic o1 still regards the plan to be true.

6 Conclusion

In our paper, we present a method for generating scalable PDDL (Planning Domain Definition Language) domains automatically through the test-time scaling of large language models (LLMs). Our experiments show that LLMs can outperform closed-source state-of-the-art models, such as OpenAI's o1-mini model, by leveraging test-time scaling, in synthesizing PDDL domains. We explore two strategies: Best-of-N (BoN) and Verbalized Machine Learning (VML). Our experiments indicate that VML achieves superior outcomes compared to BoN due to its iterative learning process, which refines solutions through self-critique feedback. By using BoN sampling to initialize solutions, we further enhance VML's performance, leading to quicker convergence and more optimal results. Our method allows for effective generalization in generating PDDL problems without the need for additional training. Consequently, it produces superior results relative to previous methods that have been extensively trained on PDDL problem synthesis. When applied to planning scenarios, our approach generates high-quality PDDL domains capable of addressing intricate planning challenges with the aid of search algorithms like A$*$. Utilizing formal languages like PDDL as planning abstractions enables

search algorithms to construct state transition graphs explicitly. In contrast, directly employing LLMs as planners can lead to inaccuracies in estimating state transitions, resulting in rule violations.

7 Limitations and Broader Impact Statement

Our work is partially motivated by the idea of program-centric abstraction introduced in the talk[2]. The idea is to learn a discrete program representation of the problem and propose reasoning on that representation. The advantages of such a method are: 1. Compared with natural language, code allows convenient verifications via compiler. 2. The program can represent states and state transitions formally than natural language. For instance, in Lean, the formal language used for mathematical proofs, it is possible to determine whether a proof goal has been achieved explicitly. This is facilitated by tactics, which are program components in Lean that formally represent proof strategies and model how goals evolve when specific tactics are applied. In our cases, PDDL uses first-order logic to represent the states and can use a logic solver *e.g.*SAT [DP60] to check whether the agent reaches the goal states. By combining the advantages of formal language with deep learning techniques, in our case, LLMs with test-time scaling can resolve the problem of combinatorial explosion in traditional program search, and prevent LLMs from hallucination during long-horizon reasoning and planning. The limitations of our work are that:

1. Using FoL for state representation aligns with PDDL's capabilities but restricts expressiveness compared to higher-order logic (HOL), which PDDL cannot handle.

2. We have yet to evaluate our methods in scenarios characterized by highly complex action spaces, such as those encountered in mathematical theorem proving.

3. The VML method requires numerous iterations to converge on effective solutions, pointing to a need for improving VML efficiency in future research.

[2]https://youtu.be/s7_NlkBwdj8

References

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. 3

[BG20] Blai Bonet and Hector Geffner. Learning first-order symbolic representations for planning from the structure of the state space. In *ECAI 2020*, pages 2322–2329. IOS Press, 2020. 3

[DJP+24] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 7

[DMAM24] Nicola Dainese, Matteo Merler, Minttu Alakuijala, and Pekka Marttinen. Generating code world models with large language models guided by monte carlo tree search. *arXiv preprint arXiv:2405.15383*, 2024. 3

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960. 12

[FN71] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 189–197, Menlo Park, California, 1971. Stanford Research Institute. 16

[GVSK23] 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 *Advances in Neural Information Processing Systems 36 (NeurIPS 2023) Main Conference Track*, 2023. 2, 3, 4

[GZY+24] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024. 9

[HGM+23] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*, 2023. 3

[JSM+23] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023. 10

[Kam24] Subbarao Kambhampati. Can large language models reason and plan? *Annals of the New York Academy of Sciences*, 1534(1):15–18, 2024. 2

[McD00] Drew M. McDermott. The 1998 ai planning systems competition. *AI Magazine*, 21(2):35, Jun. 2000. 3

[Mir24] Seyed Iman Mirzadeh. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, October 2024. 2

[TKE24] Hao Tang, Darren Key, and Kevin Ellis. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment. *arXiv preprint arXiv:2402.12275*, 2024. 3

[TMH+24] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024. 10

[VMO+24] Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36, 2024. 2

[VSK24] Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. Llms still can't plan; can lrms? a preliminary evaluation of openai's o1 on planbench. *arXiv preprint arXiv:2409.13373*, 2024. 2

[WLB+24] Kevin Wang, Junbo Li, Neel P Bhatt, Yihan Xi, Qiang Liu, Ufuk Topcu, and Zhangyang Wang. On the planning abilities of openai's o1 models: Feasibility, optimality, and generalizability. *arXiv preprint arXiv:2409.19924*, 2024. 2

[WS23] Jason Weston and Sainbayar Sukhbaatar. System 2 attention (is something you might need too). *arXiv preprint arXiv:2311.11829*, 2023. 5, 8

[XBSL24] Tim Z Xiao, Robert Bamler, Bernhard Schölkopf, and Weiyang Liu. Verbalized machine learning: Revisiting machine learning with language models. *arXiv preprint arXiv:2406.04344*, 2024. 2, 5, 6

[YCL+24] Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, et al. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*, 2024. 7

[YXWK24] Ping Yu, Jing Xu, Jason Weston, and Ilia Kulikov. Distilling system 2 into system 1. *arXiv preprint arXiv:2407.06023*, 2024. 3

[YYH+24] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024. 7

[Z+24] Hugh Zhang et al. Planning in natural language improves llm search for code generation. *arXiv preprint arXiv:2409.03733*, 2024. 5

[ZCS+23] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023. 3

[ZHB+24] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024. 3

[ZVL+24] Max Zuo, Francisco Piedrahita Velez, Xiaochen Li, Michael L Littman, and Stephen H Bach. Planetarium: A rigorous benchmark for translating text to structured planning languages. *arXiv preprint arXiv:2407.03321*, 2024. 3, 4, 10

A Pipeline

A detailed pipeline illustration can be found in Figure A.1.

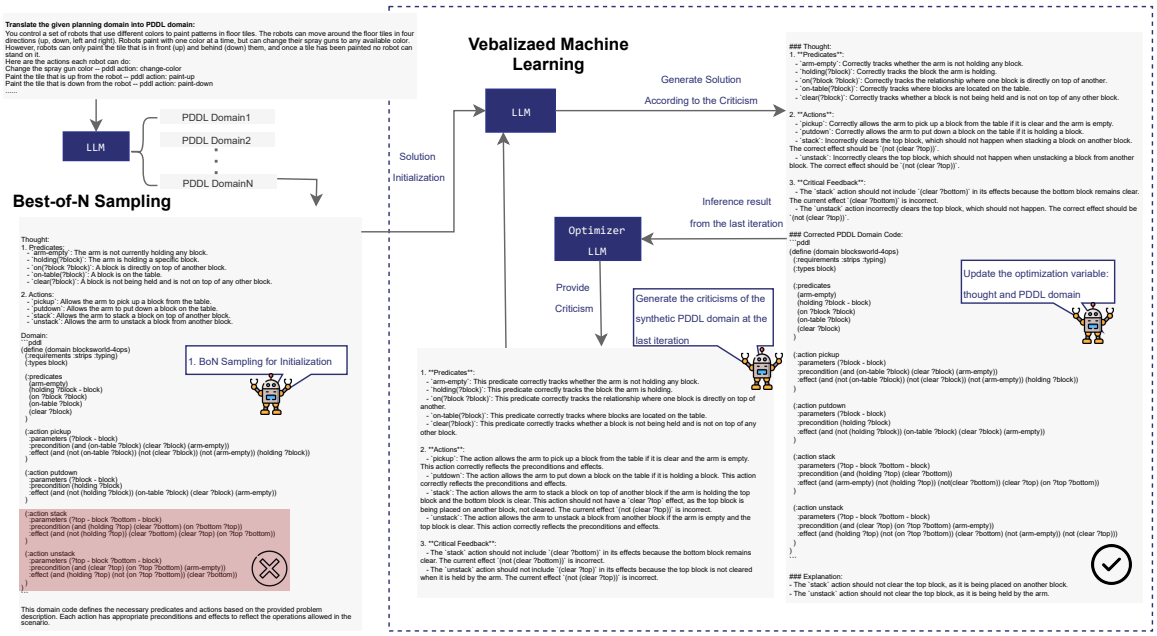

Figure A.1: Pipeline with Concrete Case Illustrations.

B Planning Problem Formulation

The classical planning problem [FN71] in artificial intelligence involves finding a sequence of actions that transition an agent from an initial state to a desired goal state within a deterministic and fully observable environment. It is formalized as a tuple $\langle \mathcal{S}, \mathcal{A}, T, s_0, G \rangle$, where: $\mathcal{S}$ is the set of all possible states; $\mathcal{A}$ is the set of all possible actions; $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the state transition function, specifying the outcome state resulting from applying an action in a state; $s_0 \in \mathcal{S}$ is the initial state; G is the goal condition, a predicate over states. The objective is to find a sequence of actions $a_1, a_2, \ldots, a_n \in \mathcal{A}$ such that applying these actions successively transitions the system from s_0 to a state s_n satisfying the goal condition G:

$$s_n = T(s_{n-1}, a_n) = T(T(\ldots T(T(s_0, a_1), a_2), \ldots, a_{n-1}), a_n)$$

with

$$s_n \models G$$

Previous works that adopt LLMs as planners estimate state transitions implicitly within language latent spaces, lacking explicit representations of the state space required for classical planning. This implicit representation can make it challenging to ensure consistency, validity, and completeness in the planning process. In contrast, our work leverages LLMs to generate explicit representations of the state space by creating PDDL domains. We utilize the generative capabilities of LLMs to produce formal PDDL models from high-level descriptions of the planning tasks. This approach bridges the gap between natural language specifications and formal planning models.

By generating PDDL domains using LLMs, we obtain: 1. explicit definitions of the set of states $\mathcal{S}$ through predicates and objects, 2. formal specifications of actions $\mathcal{A}$, including their preconditions and effects, 3. a deterministic state transition function T derived from the action definitions, 4. abilities to cooperate with clearly defined initial state s_0 and goal condition G in PDDL syntax. Thus the new objective under this background is: Given high-level descriptions of planning tasks, our objective is to leverage LLMs to create PDDL domains that can represent state transitions explicitly. By generating these explicit representations, we enable classical planning algorithms to efficiently search for plans using the defined state transitions during the planning process.

C Tasks in Case Study

Barman. The main goal of the Barman domain is to simulate the task of a bartender who prepares and serves cocktails by manipulating ingredients, tools, and glassware within a bar setting. In this scenario, the agent is tasked with creating a specific cocktail by following a series of actions that involve: 1. Identifying and dispensing the necessary ingredients required for the cocktail from the available dispensers. 2. Using bar tools such as shakers and shot glasses effectively, ensuring they are clean and suitable for use. 3. Coordinating the use of both hands to pick up and handle objects while ensuring that hands are free when needed and that objects are properly placed on surfaces like the bar counter when not in use. 4. Adhering to the proper sequence of steps for cocktail preparation, which includes dispensing ingredients into the shaker, mixing them, and then pouring the mixture into a shot glass. 5. Maintaining the correct state of all objects involved, such as keeping the shaker and glasses clean and empty before use, and updating their states appropriately as actions are performed. 6. Successfully preparing the cocktail and having it contained in the shot glass, thereby fulfilling the goal of serving the drink as intended.

Gripper. Gripper problem is designed to test the agent's ability to manage resources and plan actions in a scenario involving manipulating multiple objects across different locations. The agent, represented by one or more robots, must strategically perform the following tasks: Pick Up Balls: The agent must use its available grippers to pick up the balls from their initial locations. This requires careful hand management to ensure grippers are free and available when needed. Transport Balls: The robot needs to navigate between rooms to move balls to their specified target locations efficiently. This involves planning the correct sequence of moves and ensuring that the robot is in the correct room with the appropriate objects. Drop Balls: The agent must release the balls in the designated rooms. This requires ensuring that the robot's grippers are properly aligned and that the release actions are performed at the correct time. Manage Resources: Throughout the task, the agent must effectively manage both its grip and position within the environment, making sure that it follows constraints such as carrying capacity and room access.

Tyreworld. The main goal of the Tyreworld problem is to simulate the challenges of vehicle maintenance and tire management in a scenario where a vehicle may suffer random tire failures. The primary objectives include: 1. Replacing Flat Tires: The agent must effectively replace flat tires with intact ones on the vehicle's hubs. 2. Inflating Tires: The intact tires must be inflated before being mounted onto the vehicle. 3. Ensuring Secure Fastening: After replacing and inflating the tires, the nuts on the hubs must be securely tightened to ensure the wheels are safely attached. 3. Resource Management: The agent must manage limited resources (e.g., spare tires, tools like jacks and wrenches) strategically to minimize the risk of failure or being stranded. 4. Navigating Uncertainty: The agent must effectively plan actions while accounting for the possibility of tire failures and other uncertainties in the environment. 5. Reaching the Destination: Ultimately, the goal is to ensure the vehicle is properly equipped with intact, inflated tires, allowing it to continue

its journey successfully. 6. The Tyreworld problem serves to test an agent's planning, resource management, and adaptability in unpredictable scenarios related to vehicle maintenance.

Floor-tile. The main goal of Floor-tile is to enable the robot to navigate the environment, manage its colors, and paint the tiles according to specific requirements. The robot must efficiently utilize its movements and actions to achieve its painting objectives while adhering to the constraints of tile occupancy and color availability. In Floor-tile, three object types are defined: robot, tile, and color. Initially, the robot is located on a specific tile while holding a color. It can move up, down, right, or left with different costs: moving up costs 3, while moving down, right, or left costs 1. The robot can paint a tile above or below for a cost of 2, and changing its color incurs a cost of 5.

Termes. In Termes, a robot operates in an environment to manage and manipulate blocks at different positions. The robot can perform several actions, including moving between adjacent positions, placing blocks onto stacks, removing blocks from stacks, and creating or destroying blocks at designated depot locations. Each position on the grid has a specific height, and the robot must respect these height constraints when moving or manipulating blocks. The robot can only carry one block at a time, and it must be at the same height level to move horizontally or at a height difference of one to move vertically. The goal is to efficiently use these capabilities to achieve specific block arrangements or configurations within the environment, adhering to the constraints of adjacency, height, and block availability.

D Generated Domains

Barman

Domain.

```
(define (domain barman)
  (:requirements :strips :typing)
  (:types hand level beverage dispenser container - object ingredient cocktail - beverage
  shot shaker - container)
  (:predicates  (ontable ?c - container)
                (holding ?h - hand ?c - container)
                (handempty ?h - hand)
                (empty ?c - container)
                (contains ?c - container ?b - beverage)
                (clean ?c - container)
                (used ?c - container ?b - beverage)
                (dispenses ?d - dispenser ?i - ingredient)
                (shaker-empty-level ?s - shaker ?l - level)
                ......
  (:action grasp
             :parameters (?h - hand ?c - container)
             :precondition (and (ontable ?c) (handempty ?h))
             :effect (and (not (ontable ?c)) (not (handempty ?h)) (holding ?h ?c)))
  (:action leave ......
  (:action fill-shot ......
  (:action refill-shot ......
  (:action empty-shot ......
  (:action clean-shot ......
  (:action pour-shot-to-clean-shaker ......
  (:action pour-shot-to-used-shaker ......
  (:action empty-shaker ......
  (:action clean-shaker ......
  (:action shake ......
  (:action pour-shaker-to-shot ...... )
```

Problem.

```
(define (problem prob)
 (:domain barman)
 (:objects
      shaker1 - shaker left right - hand shot1 shot2 shot3 shot4 - shot ingredient1
      ingredient2 ingredient3 - ingredient cocktail1 cocktail2 cocktail3 - cocktail
      dispenser1 dispenser2 dispenser3 - dispenser l0 l1 l2 - level
)
 (:init
  (ontable shaker1)
  (ontable shot1) ......
  (clean shaker1) ......
  (empty shaker1) ......
  (cocktail-part1 cocktail1 ingredient1) ......
)
 (:goal
  (and (contains shot1 cocktail1) (contains shot2 cocktail3) (contains shot3 cocktail2)
```

Plan. (grasp left shot4) (fill-shot shot4 ingredient2 left right dispenser2) (pour-shot-to-clean-shaker shot4 ingredient2 shaker1 left l0 l1) (clean-shot shot4 ingredient2 left right) (fill-shot shot4 ingredient1 left right dispenser1) (pour-shot-to-used-shaker shot4 ingredient1 shaker1 left l1 l2) (refill-shot shot4 ingredient1 left right dispenser1) (leave left shot4) (grasp right shaker1) (shake cocktail3 ingredient1 ingredient2 shaker1 right left) (pour-shaker-to-shot cocktail3 shot2 right shaker1 l2 l1) (empty-shaker right shaker1 cocktail3 l1 l0) (clean-shaker right left shaker1) (leave right shaker1) (grasp left shot4) (pour-shot-to-clean-shaker shot4 ingredient1 shaker1 left l0 l1) (clean-shot shot4 ingredient1 left right) (fill-shot shot4 ingredient3 left right dispenser3) (pour-shot-to-used-shaker shot4 ingredient3 shaker1 left l1 l2) (refill-shot shot4 ingredient3 left right dispenser3) (leave left shot4) (grasp right shaker1) (shake cocktail1 ingredient1 ingredient3 shaker1 right left) (pour-shaker-to-shot cocktail1 shot1 right shaker1 l2 l1) (empty-shaker right shaker1 cocktail1 l1 l0) (clean-shaker right left shaker1) (leave right shaker1) (grasp right shot4) (pour-shot-to-clean-shaker shot4 ingredient3 shaker1 right l0 l1) (clean-shot shot4 ingredient3 right left) (fill-shot shot4 ingredient2 right left dispenser2) (grasp left shaker1) (pour-shot-to-used-shaker shot4 ingredient2 shaker1 right l1 l2) (leave right shot4) (shake cocktail2 ingredient2 ingredient3 shaker1 left right) (pour-shaker-to-shot cocktail2 shot3 left shaker1 l2 l1) ; cost = 36 (unit cost)

BlockWorld

Domain.

```
(define (domain blocksworld)
  (:requirements :strips :equality)

  (:predicates
    (clear ?x)
    (on-table ?x)
    (arm-empty)
    (holding ?x)
    (on ?x ?y))

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

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

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

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

Problem.

```
(define (problem BW-rand-12)
    (:domain blocksworld)
    (:objects b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 )
    (:init
        (arm-empty)(on-table b1)(on b2 b5)(on b3 b8)(on b4 b12)(on b5 b7)
        (on b6 b1)(on b7 b10)
        (on-table b8)(on-table b9)(on b10 b11)(on-table b11)(on b12 b9)
        (clear b2)(clear b3)(clear b4)(clear b6))
    (:goal
        (and (on b5 b10)(on b6 b12)(on b7 b4)(on b8 b3)(on b9 b2)(on b10 b8)
        (on b11 b7)(on b12 b11))))
```

Plan.

(unstack b3 b8) (putdown b3) (pickup b8) (stack b8 b3) (unstack b2 b5) (putdown b2) (unstack b4 b12) (putdown b4) (unstack b5 b7) (stack b5 b2) (unstack b7 b10) (stack b7 b4) (unstack b10 b11) (stack b10 b8) (pickup b11) (stack b11 b7) (unstack b12 b9) (stack b12 b11) (unstack b5 b2) (stack b5 b10) (unstack b6 b1) (stack b6 b12) (pickup b9) (stack b9 b2) ; cost = 24 (unit cost)

Termes

Domain.

```
(define (domain termes)
    (:requirements :typing :negative-preconditions)
    (:types
        numb - object
        position - object)
    (:predicates
        (height ?p - position ?h - numb)
        (at ?p - position)
        (has-block)
        (SUCC ?n1 - numb ?n2 - numb)
        (NEIGHBOR ?p1 - position ?p2 - position)
        (IS-DEPOT ?p - position))
    (:action move
        :parameters (?from - position ?to - position ?h - numb)
        :precondition (and (at ?from)(NEIGHBOR ?from ?to)(height ?from ?h)(height ?to ?h))
        :effect (and (not (at ?from))(at ?to) ) )
    (:action move-up
        :parameters (?from - position ?hfrom - numb ?to - position ?hto - numb)
        :precondition (and(at ?from)(NEIGHBOR ?from ?to)(height ?from ?hfrom)(height
        ?to ?hto)(SUCC ?hto ?hfrom))
        :effect (and(not (at ?from))(at ?to)))

    (:action move-down ...
    (:action place-block ...
    (:action remove-block ...
    (:action create-block ...
    (:action destroy-block ...
)
```

Problem.

```
(define (problem termes-00038-0036-4x3x3-random_towers_4x3_3_1_3)
(:domain termes)
; termes-00038-0036-4x3x3-random_towers_4x3_3_1_3
; Initial state:
;  0   0  R0D  0
;  0   0   0   0
;  0   0   0   0
; Goal state:
;  0   0   0   0
;  0   0   0   0
;  0   3   0   0
; Maximal height: 3
(:objects
    n0 - numb.....
    pos-0-0 - position.....
)
(:init
    (height pos-0-0 n0).....
    (at pos-2-0)
    (SUCC n1 n0).....
    (NEIGHBOR pos-0-0 pos-1-0).....
    (IS-DEPOT pos-2-0)
)
(:goal
(and (height pos-0-0 n0) ..... (not (has-block)))))
```

Plan.
(unstack b3 b8) (putdown b3) (pickup b8) (stack b8 b3) (unstack b2 b5) (putdown b2) (unstack b4 b12) (putdown b4) (unstack b5 b7) (stack b5 b2) (unstack b7 b10) (stack b7 b4) (unstack b10 b11) (stack b10 b8) (pickup b11) (stack b11 b7) (unstack b12 b9) (stack b12 b11) (unstack b5 b2) (stack b5 b10) (unstack b6 b1) (stack b6 b12) (pickup b9) (stack b9 b2) ; cost = 24 (unit cost)

Floor-tile

Domain.

```
(define (domain floor-tile)
  (:requirements :typing :action-costs)
  (:types robot tile color - object)

  (:predicates
    (robot-at ?r - robot ?x - tile)
    (up ?x - tile ?y - tile)
    (down ?x - tile ?y - tile)
    (right ?x - tile ?y - tile)
    (left ?x - tile ?y - tile)

    (clear ?x - tile)
    (painted ?x - tile ?c - color)
    (robot-has ?r - robot ?c - color)
    (available-color ?c - color)
    (free-color ?r - robot))

  (:functions (total-cost))
  (:action change-color
    :parameters (?r - robot ?c - color ?c2 - color)
    :precondition (and (robot-has ?r ?c) (available-color ?c2))
    :effect (and (not (robot-has ?r ?c)) (robot-has ?r ?c2) (increase (total-cost) 5)))
  (:action paint-up
    :parameters (?r - robot ?y - tile ?x - tile ?c - color)
    :precondition (and (robot-has ?r ?c) (robot-at ?r ?x) (up ?y ?x) (clear ?y))
    :effect (and (not (clear ?y)) (painted ?y ?c) (increase (total-cost) 2))
  )
  (:action paint-down...
  (:action up ...
  (:action down ...
  (:action right ...
  (:action left ...
```

Problem.

```
(define (problem p03-432)
 (:domain floor-tile)
 (:objects tile_0-1 tile_0-2 tile_0-3 ..... tile_4-1 tile_4-2 tile_4-3 - tile
           robot1 robot2 - robot
           white black - color
)
 (:init
   (= (total-cost) 0)
   (robot-at robot1 tile_2-3)
   (robot-has robot1 white)
   (robot-at robot2 tile_1-1)
   (robot-has robot2 black)
   (available-color white)
   (available-color black)
   (clear tile_0-1) .....
   (up tile_1-1 tile_0-1) .....
   (down tile_0-1 tile_1-1) .....
   (right tile_0-2 tile_0-1) .....
   (left tile_0-1 tile_0-2) .....)
 (:goal (and
    (painted tile_1-1 white)
    (painted tile_1-2 black) .....))
 (:metric minimize (total-cost)))
```

Plan. (up robot1 tile_2-3 tile_3-3) (left robot1 tile_3-3 tile_3-2) (paint-up robot1 tile_4-2 tile_3-2 white) (up robot2 tile_1-1 tile_2-1) (down robot1 tile_3-2 tile_2-2) (up robot2 tile_2-1 tile_3-1) ..... ; cost = 54 (general cost)

Tyreworld

Domain.

```
(define (domain tyreworld)
  (:types obj - object
          tool wheel nut - obj
          container hub - object)

  (:predicates (open ?x) (closed ?x) (have ?x) (in ?x ?y) (loose ?x ?y)
        (tight ?x ?y)
        (unlocked ?x) (on-ground ?x) ......
  (:action open
    :parameters (?x - container)
    :precondition (and (unlocked ?x) (closed ?x))
    :effect (and (open ?x) (not (closed ?x))))
  (:action close
    :parameters (?x - container)
    :precondition (open ?x)
    :effect (and (closed ?x) (not (open ?x))))
  (:action fetch
    :parameters (?x - obj ?y - container)
    :precondition (and (in ?x ?y) (open ?y))
    :effect (and (have ?x) (not (in ?x ?y))))
  (:action put-away ......
  (:action loosen ......
  (:action tighten ......
  (:action jack-up ......
  (:action jack-down ......
  (:action undo ......
  (:action do-up ......
  (:action remove-wheel ......
  (:action put-on-wheel ......
  (:action inflate ......
```

Problem.

```
(define (problem tyreworld-1)
(:domain tyreworld)
(:objects  wrench jack pump - tool the-hub1 - hub nuts1 - nut
    boot - container r1 w1 - wheel)
(:init (in jack boot) (in pump boot) (in wrench boot) (unlocked boot)
    (closed boot) (intact r1) (in r1 boot) (not-inflated r1) (on w1 the-
    hub1) (on-ground the-hub1) (tight nuts1 the-hub1) (fastened the-hub1))
(:goal
    (and (on r1 the-hub1) (inflated r1) (tight nuts1 the-hub1) (in w1 boot)
    (in wrench boot) (in jack boot) (in pump boot) (closed boot))))
```

Plan.
(open boot) (fetch r1 boot) (fetch wrench boot) (fetch jack boot) (loosen nuts1 the-hub1) (jack-up the-hub1) (undo nuts1 the-hub1) (remove-wheel w1 the-hub1) (put-away w1 boot) (put-on-wheel r1 the-hub1) (do-up nuts1 the-hub1) (jack-down the-hub1) (put-away jack boot) (tighten nuts1 the-hub1) (put-away wrench boot) (fetch pump boot) (inflate r1) (put-away pump boot) (close boot) ; cost = 19 (unit cost)

E Prompts for LLM-as-Planner Methods

Termes.

Prompts for o1 on Termes

Problem Description. You control a robot that can take the following actions to build complex structures.
Move from a position to another. The new position and the old position must be at the same height.
Move up from a position to another, and the height at the new position is one block higher than the old position.
Move down from a position to another, and the height at the new position is one block lower than the old position.
Place a block at a neighboring position from the robot's current position. The robot must have a block. The current height at the robot's position and the block's position must be the same. A block cannot be placed at the depot. The height at the block's position will be one block higher than the current height.
Remove a block at a neighboring position from the robot's current position. The robot must not have a block. A block cannot be removed from the depot. The current height at the robot's position must be the same as the new height at the block's position. The new height at the block's position will be one block lower than the current height.
Create a block at the depot. The robot will have the block.
Destroy a block at the depot. The robot must have a block. Now consider a planning problem. The problem description is: The robot is on a grid with 4 rows and 3 columns. pos-0-0 pos-0-1 pos-0-2 pos-1-0 pos-1-1 pos-1-2 pos-2-0 pos-2-1 pos-2-2 pos-3-0 pos-3-1 pos-3-2 The robot is at pos-2-0. The depot for new blocks is at pos-2-0. The maximum height of blocks is 3. Your goal is to build blocks so that the height at pos-1-2 is 3. Rule: You cannot have an unplaced block at the end. Examine whether you follow the rule at each step! Can you provide an optimal plan, in the way of a sequence of behaviors, to solve the problem? And what is the final optimal cost?

F State-based graph for Termes

Figure F.1 shows the planning graph that contains explicit state transition during planning for Termes.

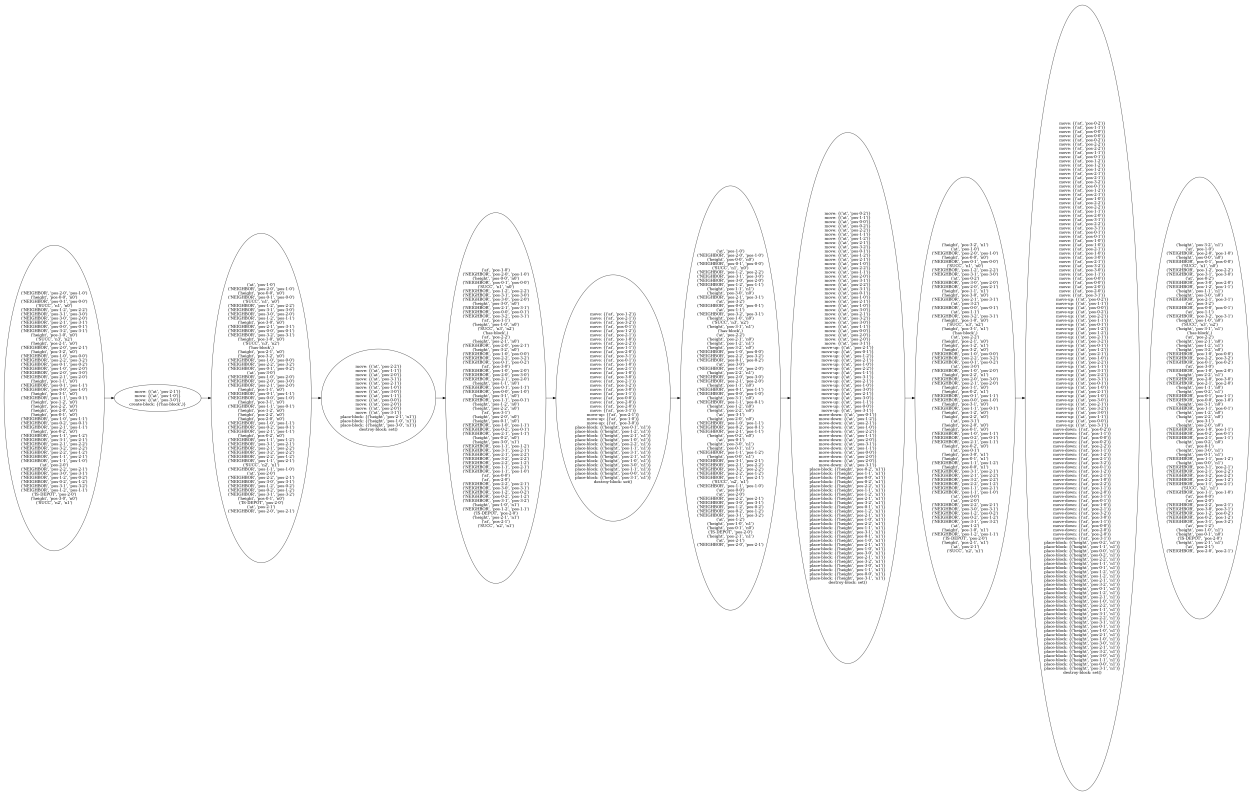

Figure F.1: The state-based graph for Termes

G The Prompt of Our Methods on Termes

Prompts for Our Methods on Termes

You will be given a natural language description of a planning problem. Your task is to translate this description into PDDL domain code. This includes defining predicates and actions based on the information provided.
Information about the AI agent will be provided in the natural language description. Note that individual conditions in preconditions and effects should be listed separately. For example, "object1 is washed and heated" should be considered as two separate conditions "object1 is washed" and "object1 is heated". Also, in PDDL, two predicates cannot have the same name even if they have different parameters. Each predicate in PDDL must have a unique name, and its parameters must be explicitly defined in the predicate definition. It is recommended to define predicate names in an intuitive and readable way.
Remember: Ignore the information that you think is not helpful for the planning task.
You are only responsible for domain generation. Before you generate the concrete domain code, you should first generate a natural language thought about the meaning of each variable, and the step-by-step explaination of the domain code. Even if I didn't provide the exact name of the predicates and actions, you should generate them based on the information provided in the natural language description.
Template is:
Thought: predicates1: the name of predicate1, explanation of predictate1 ... predicaten: the name of predicaten, explanation of predictaten action1: the name of action1, explanation of action1 ... actionn: the name of action, explanation of actionn <thought>
Domain: "'pddl The concrete pddl code for domain.pddl
Now its your time to generate the solution, you have to follow the format I provided above.
NL_Description:
You control a robot that can take the following actions to build complex structures.
Move from a position to another. The new position and the old position must be at the same height. – pddl action name: move
Move up from a position to another, and the height at the new position is one block higher than the old position. – pddl action name: move-up
Move down from a position to another, and the height at the new position is one block lower than the old position. – pddl action name: move-down
Place a block at a neighboring position from the robot's current position. The robot must have a block. The current height at the robot's position and the block's position must be the same. A block cannot be placed at the depot. The height at the block's position will be one block higher than the current height. – pddl action name: place-block
Remove a block at a neighboring position from the robot's current position. The robot must not have a block. A block cannot be removed from the depot. The current height at the robot's position must be the same as the new height at the block's position. The new height at the block's position will be one block lower than the current height. – pddl action name: remove-block
Create a block at the depot. The robot will have the block. – pddl action name: create-block
Destroy a block at the depot. The robot must have a block. – pddl action name: destroy-block
An example problem PDDL file to the domain is:
"'pddl (define (problem prob) (:domain termes) ; Initial state: ; 0 0 R0D ; 0 0 0 ; 0 0 0 ; Goal state: ; 0 0 0 ; 0 1 0 ; 0 0 0 ; Maximal height: 1 (:objects n0 - numb n1 - numb pos-0-0 - position pos-0-1 - position pos-0-2 - position pos-1-0 - position pos-1-1 - position pos-1-2 - position pos-2-0 - position pos-2-1 - position pos-2-2 - position) (:init (height pos-0-0 n0) (height pos-0-1 n0) (height pos-0-2 n0) (height pos-1-0 n0) (height pos-1-1 n0) (height pos-1-2 n0) (height pos-2-0 n0) (height pos-2-1 n0) (height pos-2-2 n0) (at pos-2-0) (SUCC n1 n0) (NEIGHBOR pos-0-0 pos-1-0) (NEIGHBOR pos-0-0 pos-0-1) (NEIGHBOR pos-0-1 pos-1-1) (NEIGHBOR pos-0-1 pos-0-0) (NEIGHBOR pos-0-1 pos-0-2) (NEIGHBOR pos-0-2 pos-1-2) (NEIGHBOR pos-0-2 pos-0-1) (NEIGHBOR pos-1-0 pos-0-0) (NEIGHBOR pos-1-0 pos-2-0) (NEIGHBOR pos-1-0 pos-1-1) (NEIGHBOR pos-1-1 pos-0-1) (NEIGHBOR pos-1-1 pos-2-1) (NEIGHBOR pos-1-1 pos-1-0) (NEIGHBOR pos-1-1 pos-1-2) (NEIGHBOR pos-1-2 pos-0-2) (NEIGHBOR pos-1-2 pos-2-2) (NEIGHBOR pos-1-2 pos-1-1) (NEIGHBOR pos-2-0 pos-1-0) (NEIGHBOR pos-2-0 pos-2-1) (NEIGHBOR pos-2-1 pos-1-1) (NEIGHBOR pos-2-1 pos-2-0) (NEIGHBOR pos-2-1 pos-2-2) (NEIGHBOR pos-2-2 pos-1-2) (NEIGHBOR pos-2-2 pos-2-1) (IS-DEPOT pos-2-0)) (:goal (and (height pos-0-0 n0) (height pos-0-1 n0) (height pos-0-2 n0) (height pos-1-0 n0) (height pos-1-1 n1) (height pos-1-2 n0) (height pos-2-0 n0) (height pos-2-1 n0) (height pos-2-2 n0) (not (has-block)))))

H Human in the Loop Experiment

We conducted human-in-the-loop experiments to refine the process of writing PDDL (Planning Domain Definition Language) domain code with the interaction between artificial intelligence and human. The experiment involved graduate students majoring in AI and robotics, focusing on evaluating the effectiveness of human-AI collaboration in generating accurate and semantically meaningful PDDL domain files that strictly adhere to given specifications.

The planning domain selected for this study was Termes, which necessitated the translation of

robot actions into PDDL. These actions included horizontal and vertical movements, block placement and removal, and depot management within a simulated environment.

In the initial phase, participants interacted directly with an AI agent by providing prompts based on descriptions of robot actions. While the AI-generated PDDL code passed basic validation, it often lacked proper definition of action preconditions or the accurate use of predicates, such as the "SUCC" predicate, which denotes an ordered relationship between items. In response to these limitations, the students refined their prompting strategy by incorporating more structured instructions and examples, resulting in improved outcomes.

Simultaneously, students manually coded PDDL domain files, utilizing the AI agent to ensure grammatical correctness. This approach facilitated the creation of logically comprehensive domain files, though several iterations were still required to achieve successful validation.

Through this iterative process, students provided critical insights into the current strengths and weaknesses of AI in comprehending complex logical structures and semantic nuances within specialized domains like PDDL. Their findings underscored the challenges associated with writing precise PDDL code and emphasized the need for an automated pipeline to facilitate PDDL domain synthesis.

Table 2: Comparison of the performance on PDDL Domain synthesis.

Model	# Params	NL2Domain	Problem2Domain	Avg.
Open-Source Models				
Qwen2.5-Instruct	0.5B	0.0	0.0	0.0
Qwen2.5-Instruct	1.5B	0.0	0.0	0.0
Qwen2.5-Instruct	3B	2.1	1.5	0.0
Qwen2.5-Instruct	7B	5.7	11.7	8.7
Qwen2.5-Instruct	14B	21.6	25.3	23.5
Qwen2.5-Instruct	32B	24.0	31.6	27.8
Qwen2.5-Instruct	72B	38.5	32.8	35.7
Qwen2.5-Coder	1.5B	0.0	0.0	0.0
Qwen2.5-Coder	7B	21.9	18.4	20.2
Llama3.1-Instruct	8B	0.0	0.0	0.0
Llama3.1-Instruct	70B	1.1	0.0	0.6
Yi-1.5-Chat	6B	0.4	1.8	1.1
Yi-1.5-Chat	9B	6.7	9.3	8.0
Yi-1.5-Chat	34B	12.0	8.7	10.4
Yi-Coder	1.5B	0.0	0.0	0.0
Yi-Coder	9B	9.9	14.5	12.2
Closed-Source Models				
GPT-4o	-	5.3	50.0	27.7
o1-mini	-	41.7	33.7	37.7
o1-preview	-	55.8	52.4	54.1
Our Methods				
BoN-8-Qwen2.5-Instrcut	0.5B	0.0 (+0.0)	0.0 (+0.0)	0.0
BoN-8-Qwen2.5-Instrcut	1.5B	2.1 (+2.1)	0.3 (+0.3)	1.2
BoN-8-Qwen2.5-Instrcut	3B	11.7 (+9.5)	1.2 (+0.3)	6.5
BoN-8-Qwen2.5-Instrcut	7B	9.2 (+3.5)	34.6 (+22.9)	21.9
BoN-8-Qwen2.5-Instrcut	14B	51.6 (+30.0)	62.0 (+36.7)	56.8
BoN-8-Qwen2.5-Instrcut	32B	66.8 (+46.7)	71.1 (+39.5)	70.9
BoN-8-Qwen2.5-Instruct	72B	60.8 (+22.3)	73.8 (+41.0)	67.3
BoN-8-Qwen2.5-Coder	7B	73.1 (+51.2)	63.3 (+44.9)	68.2
VML-5-BoN-8-Qwen2.5-Instruct	0.5B	0.0 (+0.0)	0.0 (+0.0)	0.0
VML-5-BoN-8-Qwen2.5-Instruct	1.5B	2.8 (+2.8)	0.3 (+0.3)	1.6
VML-5-BoN-8-Qwen2.5-Instruct	3B	18.7 (+16.6)	1.8 (+0.3)	10.3
VML-5-BoN-8-Qwen2.5-Instruct	7B	21.9 (+16.2)	49.1 (+37.3)	35.5
VML-5-BoN-8-Qwen2.5-Instruct	14B	77.0 (+55.4)	80.4 (+55.1)	78.7
VML-5-BoN-8-Qwen2.5-Instruct	32B	86.2 (+62.2)	90.9 (+59.3)	88.6
VML-5-BoN-8-Qwen2.5-Instruct	72B	78.4 (+39.9)	86.4 (+53.6)	82.4
VML-5-BoN-8-Qwen2.5-Coder	7B	85.2 (+63.3)	71.4 (+53.0)	78.3

Table 3: Action-level PDDL Synthesis Comparison of BoN with VML

Natural Language	**BoN@8**	**BoN@8 + VML@5**
BlockWorld(stack): The arm can stack a block on top of another if it holds the top block and the bottom block is clear. Post-action, the arm is empty, the top block rests on the bottom block, and the bottom block is no longer clear.	(:action stack :parameters (?top - block ?bottom - block) :precondition (and (holding ?top) (clear ?bottom)) :effect (and (not (holding ?top)) (on ?top ?bottom) (not (clear ?bottom))))	Incorrectly clear the bottom block, which should not happen when stacking a block on the bottom block. (:action stack :parameters (?top - block ?bottom - block) :precondition (and (holding ?top) (clear ?bottom)) effect (and (arm-empty) (not (holding ?top)) (not (clear ?bottom)) (on ?top ?bottom)))
BlockWorld(unstack): allows the arm to unstack a block from on top of another block if the arm is empty and the top block is clear. After the unstack action, the arm will be holding the top block, the top block will no longer be on top of the bottom block, and the bottom block will be clear.	(:action unstack :parameters (?top - block ?bottom - block) :precondition (and (clear ?top) (on ?top ?bottom) (arm-empty)) :effect (and (holding ?top) (not (on ?top ?bottom)) (clear ?bottom)))	The unstack action should not include (clear?top) in its effects because the top block is not cleared when it is held by the arm. (:action unstack :parameters (?top - block ?bottom - block) :precondition (and (clear ?top) (on ?top ?bottom) (arm-empty)) :effect (and (holding ?top) (not (on ?top ?bottom)) (clear ?bottom) (not (arm-empty))))
TSP(return-against): This action also completes the tour by allowing the agent to return to the starting city against the direction of a connection. The precondition is similar to the "return-along" action, except the city is not previously visited. The effect is that the agent returns to the starting point and the tour is marked as complete.	(:action return-against :parameters (?current-city - city ?start-city - city) :precondition (and (in ?current-city) (connected ?start-city ?current-city)(not-complete)) :effect (and (in ?start-city) (complete)))	The precondition for this action should have "(connected ?current-city ?start-city)" instead of "(connected ?start-city ?current-city)" to correctly reflect the travel direction. (:action return-against :parameters (?current-city - city ?start-city - city) :precondition (and (in ?current-city) (connected ?current-city ?start-city) (not-complete)) :effect (and (in ?start-city) (complete)))
Termes(remove-block): Create a block at the depot. The robot will have the block	(:action remove-block :parameters (fromPos toPos) :precondition (and (height fromPos ?h) (SUCC ?h ?h-down) (height toPos ?h-down) (at fromPos) (not (has-block))) :effect (and (has-block) (height toPos ?h)))	The effect of the "remove-block" action does not correctly update the height of the block's position. It should be "(height toPos ?h-down)", but it is currently "(?h-down)". (:action remove-block :parameters (fromPos toPos) :precondition (and (height fromPos ?h) (SUCC ?h ?h-down) (height toPos ?h-down) (at fromPos) (not (has-block))) :effect (and (has-block) (height toPos ?h-down)))

Table 4: Performance Comparison of Different Models on PDDL problem generation

Model	**Setting**	**Correct Rate (%)**
Gemma 1.1 IT 2B	Zero-shot	0.00
	Fine-tuned	94.21
Gemma 1.1 IT 7B	Zero-shot	0.00
	Fine-tuned	98.79
Mistral v0.3 Instruct 7B	Zero-shot	0.01
	Fine-tuned	99.00
GPT-4o	Zero-shot	35.12
Ours (Qwen2.5-Coder-7B)	BoN@16	99.24
Ours (Qwen2.5-Coder-7B)	BoN@16 + VML@1	99.60

	Setting	Floortile	Barman	Tyreworld	Grippers	Termes	Blockworld
LLM-as-Planner Methods							
gpt-4o	pass@1	0.0	6.7	0.0	23.8	4.8	4.8
	self-critique	10.0	13.3	0.0	33.3	0.0	14.2
	pass@8	13.3	33.3	45.0	45.0	10.0	23.8
o1-mini	pass@1	5.3	33.3	50.0	57.1	23.8	38.1
	self-critique	5.3	33.3	35.0	61.9	23.8	47.6
	pass@8	0.0	33.3	70.0	61.9	52.4	23.1
o1-preview	pass@1	0.0	13.3	33.3	38.1	0.0	4.7
	self-critique	5.0	6.7	35.0	33.3	4.7	9.5
	pass@8	33.3	33.3	85.0	66.7	19.0	33.3
Our Methods (PDDL as Abstraction)							
Qwen2.5-7B-Coder	BoN@4	0.0	0.0	0.0	100.0	81.0	9.5
	BoN@16	100.0	100.0	100.0	100.0	100.0	0.0
	BoN@4+VML@5	0.0	100.0	100.0	100.0	100.0	71.4
	BoN@16+VML@5	100.0	100.0	100.0	100.0	100.0	81.0

Table 5: Comparison between our methods using PDDL as abstraction with LLM-as-Planner methods.