Planning in the Dark: LLM-Symbolic Planning Pipeline Without Experts

Sukai Huang[1], **Nir Lipovetzky**[1] **and Trevor Cohn**[1,2*]

[1]The University of Melbourne
[2] Google
sukaih@student.unimelb.edu.au, {nir.lipovetzky, trevor.cohn}@unimelb.edu.au

Abstract

Large Language Models (LLMs) have shown promise in solving natural language-described planning tasks, but their direct use often leads to inconsistent reasoning and hallucination. While hybrid LLM-symbolic planning pipelines have emerged as a more robust alternative, they typically require extensive expert intervention to refine and validate generated action schemas. It not only limits scalability but also introduces a potential for biased interpretation, as a single expert's interpretation of ambiguous natural language descriptions might not align with the user's actual intent. To address this, we propose a novel approach that constructs an action schema library to generate multiple candidates, accounting for the diverse possible interpretations of natural language descriptions. We further introduce a semantic validation and ranking module that automatically filter and rank the generated schemas and plans without expert-in-the-loop. The experiments showed our pipeline maintains superiority in planning over the direct LLM planning approach. These findings demonstrate the feasibility of a fully automated end-to-end LLM-symbolic planner that requires no expert intervention, opening up the possibility for a broader audience to engage with AI planning with less prerequisite of domain expertise.

Code — https://github.com/Sino-Huang/Official-LLM-Symbolic-Planning-without-Experts

Extended version — https://arxiv.org/abs/2409.15922

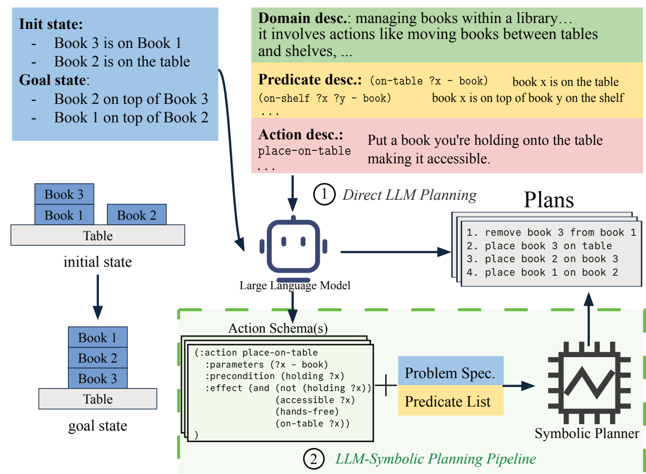

Figure 1: An overview of direct plan generation vs. LLM-symbolic planning pipelines.

1 Introduction

The advent of Large Language Models (LLMs) has opened new avenues for solving natural language-described planning tasks (Kojima et al. 2022). However, direct plan generation using LLMs, while seemingly straightforward, has been criticized for inconsistent reasoning and hallucination, which undermines their reliability in critical planning scenarios (Valmeekam et al. 2022, 2023; Huang et al. 2024). In response, researchers have advocated for more robust approaches that combine the flexibility of LLMs with the correctness of symbolic planning to solve planning tasks (Pallagani et al. 2024; Oswald et al. 2024). To improve the soundness of generated plans, a hybrid LLM-symbolic planning pipeline has emerged. As shown in Figure 1, instead of relying solely on LLMs to generate sequences of action plans through in-context learning, this pipeline begins by leveraging LLMs to extract abstract symbolic action specifications from natural language descriptions, known as *action schemas*. These schemas define the essential components of an action in a structured format understandable by symbolic planners. Once these schemas are generated, a classical planner can take over to search for feasible plans that fulfill the task specifications (Liu et al. 2023; Silver et al. 2024; Guan et al. 2023; Kambhampati et al. 2024).

Yet, this method is brittle, as a single missing or contradictory predicate in an action schema can prevent the planner from finding a valid plan. Thus, current pipelines often require multiple iterations of expert intervention to refine and validate the generated action schemas. For instance, Guan et al. (2023) reported that the expert took 59 iterations to fix schema errors for a single task domain. This process demands substantial time and expertise, which significantly hinders the ***scalability*** of the method. More critically, due to budget constraints, often only one expert is involved in the process. This creates a critical vulnerability: the potential for interpretation mismatch between the expert and the user. Ex-

*Now at Google DeepMind

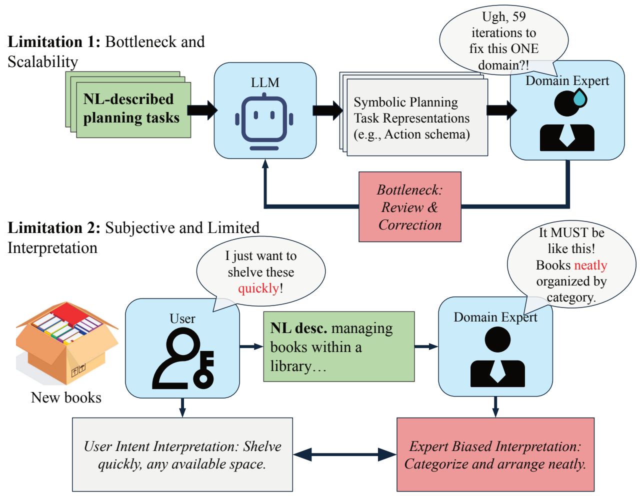

Figure 2: Illustration of the two limitations of expert-dependent LLM-symbolic planning pipelines

perts, while knowledgeable, inevitably bring their own ***subjective interpretations*** to the task descriptions, often formalizing them in a single, specific way. This limits the system to a ***single perspective*** of the task. However, unlike formal language designed to have an exact, context-independent meaning, natural language inherently contains ambiguities that yield *diverse* valid interpretations of the same description. This ambiguity suggests that a straightforward, one-to-one mapping from natural to formal languages – a typical case when relying on a single expert – risks overlooking the interpretation that the user actually intended (Moravcsik 1983) (see Figure 2).

Regarding the issue with reliance on expert intervention, we propose a novel pipeline that eliminates this dependency. Specifically, our approach introduces two key innovations:

(1): We construct an *action schema library* to generate multiple candidates, a strategy that has been overlooked in prior work despite being a natural fit for capturing the inherent ambiguity in natural language. By leveraging this library, we also increase the likelihood of obtaining ***solvable*** action schema sets – those have at least one valid plan that can be found by a planner.

(2): We leverages sentence encoders[1] to automatically *validate and filter generated action schemas*. This module ensures that the generated schemas closely align with the task descriptions in the semantic space, effectively acting like expert feedback. Our experiments demonstrate that without expert intervention, our pipeline generates sound action plans competitive with direct LLM-based plan generation, even in short-horizon planning tasks. Importantly, our approach offers multiple schema sets and plan candidates, preserving the diversity of interpretations inherent in ambiguous natural language descriptions.

2 Related Work

Direct Plan Generation with LLMs: The use of LLMs for direct action plan generation has been explored across various domains, including embodied tasks (Wang et al. 2023; Xiang et al. 2024), and other language grounding environments (Ahn et al. 2022; Huang et al. 2022). These approaches are built upon the idea that LLMs' reasoning capabilities can be effectively elicited through in-context learning techniques, particularly the Chain-of-Thought (CoT) approach. CoT prompts the model to generate a series of intermediate reasoning steps before arriving at the final answer, resulting in more coherent and logically sound reasoning (Wei et al. 2022). Building upon CoT, Yao et al. (2024) proposed Tree-of-Thought (ToT) framework, which explores multiple reasoning pathways, generating diverse plans and ranking them based on self-verification heuristics. These heuristics are verbalized confidence scores produced by LLMs themselves, a method supported by studies showing that LLMs are effective as zero-shot ranking models (Lin et al. 2022; Hou et al. 2023; Zhuang et al. 2023).

Criticism and Hybrid Planning: Despite the promising results, researchers have raised concerns about the reliability and soundness of LLM-generated plans (Valmeekam et al. 2022, 2023; Huang et al. 2024). A critical issue highlighted by Kambhampati et al. (2024) is that planning and reasoning tasks are typically associated with System 2 competency, which involves slow, deliberate, and conscious thinking (Sloman 1996; Kahneman 2011). However, LLMs, being essentially text generators, exhibit constant response times regardless of the complexity of the question posed. This behavior suggests that no first-principle reasoning is occurring, contradicting the expectations for true planning capabilities. To this end, researchers have explored hybrid approaches. For instance, Thought of Search (Katz et al. 2024) involves the generation of successor function and goal test code by LLMs, followed by their execution within an external execution environment. The approach we focus on involves utilizing LLMs to generate symbolic representations of tasks, which are then processed by external symbolic planners to search for feasible plans (Liu et al. 2023; Guan et al. 2023). However, existing pipelines emphasize the necessity of expert intervention for action schema validation and refinement. While Kambhampati et al. (2024) proposed using LLMs as semi-expert critics to assess output quality, this approach still necessitates expert involvement for final decision-making. In contrast, our work strives to reduce the dependency on expert intervention, offering a more accessible approach to hybrid LLM-symbolic planning that also addresses the inherent ambiguity in natural language descriptions.

3 Problem Setting and Background

We consider a scenario where an agent generates action plans for natural language-described planning tasks. A task description typically consists of: (1) a domain description outlining general task information and possible high-level

[1]Sentence encoders are neural network models that transform sentences into vector representations, capturing semantic meaning

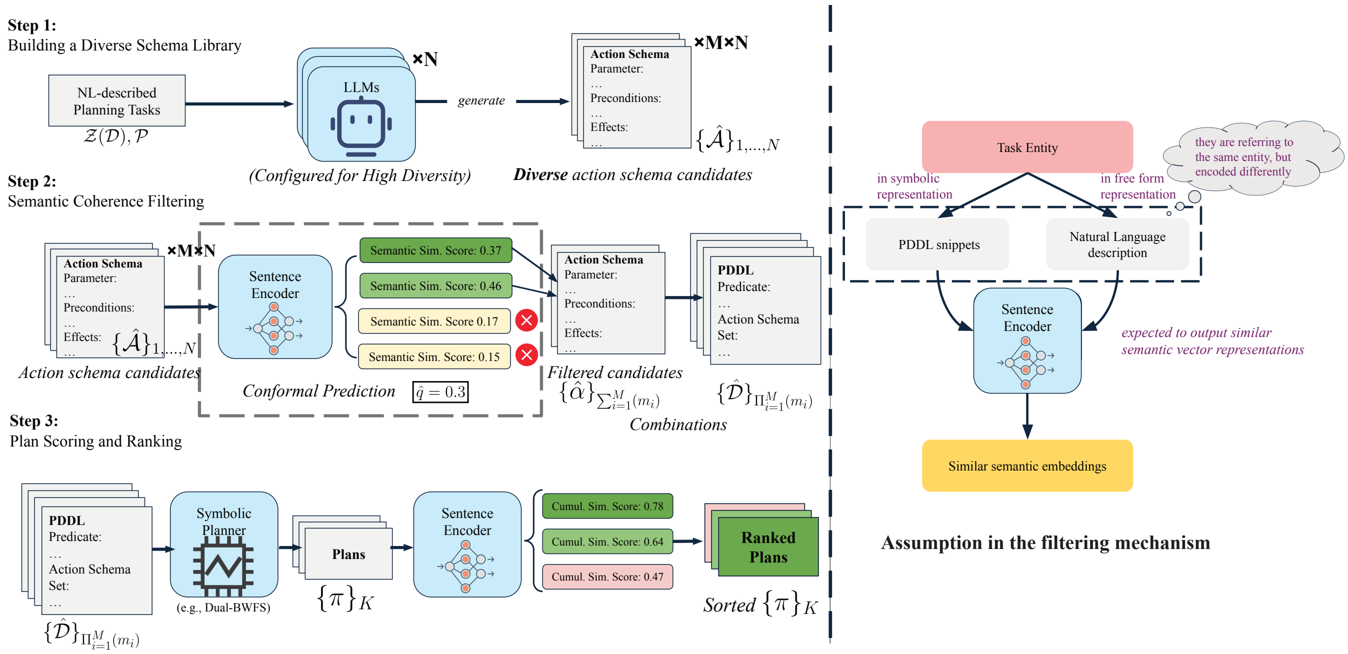

Figure 3: An overview of the proposed pipeline, it first constructs diverse action schema candidates to cover various interpretations of the natural language descriptions. Then, it filters out low-confidence candidates to ensure the generation candidates are semantically aligned with the descriptions. Lastly, it produces and ranks multiple plans using a symbolic planner. The filtering mechanism is grounded in the concept of semantic equivalence across different representations of the same content.

actions, and (2) a problem instance description specifying the initial and goal states. The study of LLM-symbolic planning pipelines is grounded in the formal framework of classical planning, which relies on symbolic representations of planning tasks. These representations are typically expressed using the Planning Domain Definition Language (PDDL) (Aeronautiques et al. 1998; Haslum et al. 2019). In brief, a PDDL description is defined by $\langle \mathcal{D}, \Pi_{\mathcal{D}} \rangle$, where:

- $\mathcal{D} = \langle \mathcal{P}, \mathcal{A} \rangle$ is the domain specification: $\mathcal{P}$ is the set of predicates that can either hold true or false, and $\mathcal{A}$ is the set of action schemas. Each action schema $\alpha \in \mathcal{A}$ is defined as a tuple $\alpha = \langle par, pre, eff \rangle$, where par details the parameters, and pre and eff are the preconditions and effects, respectively. Both pre and eff are typically expressed as conjunctive logical expressions using predicate logic.
- $\Pi_{\mathcal{D}} = \langle \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ is the problem instance: $\mathcal{O}$ is the set of objects to interact with, $\mathcal{I}$ is the initial state, and $\mathcal{G}$ is the goal state that the agent needs to achieve.

A solution to the planning task is a sequence of grounded actions ($\pi = (a_0, ..., a_n)$) that transforms the initial state $\mathcal{I}$ to the goal state $\mathcal{G}$. Each grounded action a_i is an instantiation of an action schema $\alpha \in \mathcal{A}$ and predicates, where the parameters in α are replaced with specific objects from $\mathcal{O}$.

To bridge natural language descriptions and formal planning representations, we introduce a natural language proxy layer, denoted as $\mathcal{Z}(\cdot)$, for these task specifications. For example, $\mathcal{Z}(\mathcal{D})$ represents the natural language equivalent of the domain specification $\mathcal{D}$. The two approaches, *direct LLM planning* and *LLM-symbolic planning*, can then be expressed in Eq 1 and Eq 2, respectively:

$$\pi \sim P_{\text{LLM}}(\cdot \mid \mathcal{Z}(\mathcal{D}), \mathcal{Z}(\Pi_{\mathcal{D}})) \tag{1}$$

$$\hat{\mathcal{A}} \sim P_{\text{LLM}}(\cdot \mid \mathcal{Z}(\mathcal{D}))\,; \Pi_{\mathcal{D}} \sim P_{\text{LLM}}(\cdot \mid \mathcal{Z}(\Pi_{\mathcal{D}}))$$
$$\pi = f\left(\langle \mathcal{P}, \hat{\mathcal{A}} \rangle, \Pi_{\mathcal{D}}\right) \tag{2}$$

In these equations, $P_{\text{LLM}}(\cdot)$ represents the generation process of LLMs, and f is the symbolic planner that search for sound plans. While we largely adhere to the problem setting of previous research (e.g., Liu et al. (2023), Guan et al. (2023)), we introduce a crucial refinement by specifying a precise predicate set ($\mathcal{P}$) for each domain descriptions. This controlled setting addresses a key challenge in evaluating across different methodologies. Without a standardized predicate set, variations in domain understanding can lead to diverse and potentially incomparable outputs, hindering meaningful evaluation.

4 Methodology

As illustrated in Figure 3, the proposed pipeline stands in contrast to existing expert-dependent approaches and consists of three key steps: *(1) Building a Diverse Schema Library (§4.1), (2) Semantic Coherence Filtering (§ 4.2) and (3) Plan Scoring and Ranking (§ 4.4).*

4.1 Building a Diverse Schema Library

A key challenge in translating natural language descriptions into symbolic action schemas is the inherent ambiguity of language itself. Different interpretations of the same description can lead to variations in action schemas, impacting the downstream plan generation process. To ensure we explore a wide range of interpretations and effectively cover the user's intent, we utilize multiple LLM instances, denoted as $\{P^1_{\text{LLM}}, P^2_{\text{LLM}}, ..., P^N_{\text{LLM}}\}$, and set their temperature hyperparameter high to encourage diverse outputs. Each will then generate its own set of action schemas $\hat{\mathcal{A}}_i \sim P^i_{\text{LLM}}(\cdot \mid \mathcal{Z}(\mathcal{D}))$, where $\hat{\mathcal{A}}_i = (\hat{\alpha}_{i1}, \hat{\alpha}_{i2}, ..., \hat{\alpha}_{iM})$. Here, $\hat{\alpha}_{ij}$, where $i \in [1, ..N]$ and $j \in [1, ..., M]$, represents the generated action schema of j-th action in the domain by the i-th LLM instance.

The generated schemas $\hat{\alpha}_{ij}$ from all models are then aggregated into a single library. Since each domain comprises M actions, a "set" of action schemas refers to a complete collection where each action in the domain is associated with one corresponding schema. Therefore, all possible combination of action schemas within the library can generate approximately $\binom{N}{1}^M$ different sets of action schemas.

In addition, existing pipelines rely heavily on expert intervention, partly because individual LLMs struggle to generate *solvable* sets of schemas – those that a planner can successfully use to construct a plan. This reliance becomes even more pronounced as the number of actions increases, with the probability of obtaining a solvable set of schemas from a single LLM diminishing exponentially. In contrast, our approach, by constructing a diverse pool of action schema sets, substantially improves the probability of finding a solvable set. Our analysis (detailed in Appendix A) demonstrates that, under reasonable assumptions, this probability can increase from less than 0.0001% with a single LLM to over 95% when using multiple LLM instances.

Note that the *solvability* of a set of action schemas can be efficiently verified by leveraging the *completeness* feature of modern symbolic planners. If a plan can be found for a given problem using the generated schemas, the set is deemed solvable. Importantly, modern symbolic planners have advanced capabilities that allow them to efficiently reject unsolvable schema sets. This is achieved by the ability to prove delete-free reachability in polynomial time (Bonet and Geffner 2001). Furthermore, modern planners are designed to operate efficiently on multithread CPU and the efficiency of the process should not be a cause for concern. See Appendix D for more details.

4.2 Semantic Coherence Filtering

The previous method alone faces two limitations. First, as task complexity grows, the "brute-force" approach of combining and evaluating all possible sets becomes increasingly inefficient. Second, solvability does not guarantee semantic correctness – schemas may not accurately reflect the task descriptions, potentially leading to incorrect or nonsensical plans. Therefore, it is crucial to implement a filtering mechanism that autonomously assesses the semantic correctness of individual action schemas, filtering out low-quality candidates before they enter the combination process.

Our approach is grounded in the concept of semantic equivalence across different representations of the same content, as discussed by Weaver (1952) in his memorandum "Translation." Weaver emphasized that the most effective way to translate between languages is to go deeper to uncover a shared "common base of meaning" between language representations, illustrating this by noting that "a Russian text is really written in English, but it has been encoded using different symbols." This principle is crucial in our context, where task descriptions in natural language and their corresponding structured symbolic representations should exhibit high semantic similarity, reflecting the same shared meaning despite different syntactic forms (see right side of Figure 3).

Recent developments in language models as code assistants (Chen, Tworek et al. 2021; Rozière, Gehring et al. 2024) further support this assumption, demonstrating that these models can decode the underlying semantics of structured symbolic representations. Inspired by this, we propose a filtering step that leverages a sentence encoder $E(\cdot)$ to generate embeddings for both the action descriptions $E(\mathcal{Z}(\alpha))$ and the generated schemas $E(\hat{\alpha})$. Then, we compute the cosine similarity between these embeddings to quantify semantic relatedness and filter out action schemas with low scores.

Specifically, we employ a conformal prediction (CP) framework (see Appendix B) to statistically guarantee that true positive action schema candidates have a high probability of being preserved while minimizing the size of the filtered set (Sadinle et al. 2019). In this process, a threshold $\hat{q}$ will be calculated based on a user-specified confidence level $1 - \epsilon$. Action schemas with cosine similarity scores below this threshold are filtered out from the library.

This process (illustrated in *step 2* of Figure 3) significantly reduces the *number of candidate sets of action schemas* to $\Pi^M_{i=1}(m_i)$, where m_i is the number of action schemas that pass the semantic validation for the i-th action. This prefiltering approach not only reduces the computational load on the symbolic planner, increasing efficiency, but also ensures that generated schemas closely align with the semantic meaning of the task descriptions.

4.3 Finetuning with Manipulated Action Schemas

Hard negative samples have been shown to enhance representation learning by capturing nuanced semantic distinctions (Robinson et al. 2023). In our context, we found that structured action schemas are particularly ideal for generating hard negatives. By manipulating predicates in the precondition or effect expressions of true action schemas, we create hard negatives with subtle differences. During finetuning, a triplet loss function is employed, where each training sample consists of a triplet: the natural language description of an action $(\mathcal{Z}(\alpha))$, the true action schema (α), and a negative sample (α^{neg}). A negative sample is of three types – *(1) Easy Negatives*: action schemas from other planning domains (inter-domain mismatch); *(2) Semi-Hard Negatives*: action schemas from the same domain but referring to different actions (intra-domain mismatch); and *(3) Hard Negatives*: As shown in Table 1, we employ four types of manipulations – swap, negation, removal, and addition – to manipulate the reference[2] action schema of the domain.

Through this process, the sentence encoder learns to embed natural language descriptions closer to their corresponding action schemas while distancing them from negative samples in the semantic space.

4.4 Plan Generation and Ranking

Action schemas that more accurately represent the intended tasks described in natural language are likely to yield higher-quality, more reliable plans. Leveraging this causal relationship, we assess and rank the generated plans based on the cumulative semantic similarity scores of their constituent action schemas. Specifically, we feed each solvable set of action schemas into a classical planner, which generates a corresponding plan. Then, the ranking score for a plan is calculated as $\sum_{i=1}^{M} \frac{E(\mathcal{Z}(\alpha_i)) \cdot E(\hat{\alpha_i})}{\|E(\mathcal{Z}(\alpha_i))\| \|E(\hat{\alpha_i})\|}$, where $\mathcal{Z}(\alpha_i)$ is the natural language description of the i-th action in the domain, $\hat{\alpha}_i$ is the corresponding generated action schema and $E(\cdot)$ is already defined in Sec 4.2. It ensures that the structured symbolic model comprising the plans are semantically aligned with the descriptions of the planning domain (see *step 3* in Figure 3). Furthermore, this approach allows for optional **lightweight expert intervention** as a final, non-iterative step. By presenting the ranked schema sets and their corresponding plans, experts can determine the most appropriate one, providing a balance between autonomy and expert guidance.

Overall, our pipeline bridges the gap between ambiguous task descriptions and the precise requirements of symbolic planners. By generating a diverse pool of action schemas and leveraging semantic similarity for validation and ranking, we achieve two key advancements. First, we reduce the dependency on expert intervention, making the process more accessible and efficient. Second, we preserve the inherent ambiguity of natural language, offering users multiple valid interpretations of the task and their corresponding plans.

5 Experiments

Our experiments test the following hypotheses: **(H1)** Semantic equivalence across different representations, as discussed by Weaver, holds true in our context. **(H2)** Ambiguity in natural language descriptions leads to multiple interpretations. **(H3)** Our pipeline produces multiple solvable candidate sets of action schemas and plans without expert intervention, providing users with a range of options. **(H4)** Our pipeline outperforms direct LLM planning approaches in plan quality, demonstrating the advantage of integrating LLM with symbolic planning method. See Appendix for other experiments outside the scope of these hypotheses.

5.1 Experimental Setup

Task and Model Setup. We introduces several key enhancements that distinguish it from previous work. *(1) Novel Test Domains:* We carefully selected three test domains ensuring they are unfamiliar to LLMs – **Libraryworld**: a modified version of the classic Blockworld domain; **Minecraft**: resource gathering and crafting domain inspired by the game Minecraft; and **Dungeon**: a domain originally proposed by Chrpa et al. (2017). This approach addresses a significant issue: many IPC[3] domains have likely been leaked into LLM training data (see Appendix C). For training and calibration of the sentence encoder, we used domains from IPC and PDDLGym (Silver and Chitnis 2020). *(2) LLM Selection:* We use the open-source GLM (Hou et al. 2024) over proprietary models like GPT-4, aligning with our commitment to accessible planning systems. *(3) Ambiguity Examination:* We tested our pipeline on two types of task descriptions to assess the impact of ambiguity – (a) ***detailed*** descriptions following the established style of Guan et al. (2023), and (b) ***layman*** descriptions provided by five non-expert participants[4] who, unfamiliar with PDDL, described the domains and actions based on reference PDDL snippets. *(4) Symbolic Planner:* We used *DUAL-BWFS* (Lipovetzky and Geffner 2017) planner for plan generation as well as checking if the generated schema sets are solvable. *(5) LLM Prompt Engineering:* We use the CO-STAR and CoT framework to guide LLMs in generating outputs (see Appendix E).

Baselines. We evaluate our pipeline against two key baselines: *(1) The previous LLM-symbolic planning pipeline* proposed by Guan et al. (2023), which involves expert intervention for action schema validation and refinement; and *(2) Direct LLM-based planning* using Tree-of-Thought (ToT) (Yao et al. 2024), which generates multiple plans and ranks them based on self-verification heuristics.

5.2 Semantic Equivalence Analysis

To investigate **H1**, we initially assessed the cosine similarity of sentence embeddings for pairs of action schemas and their corresponding natural language descriptions, both when they were matched and when they were mismatched. We employed two pre-trained, extensive sentence encoders: *text-embedding-3-large* and *sentence-t5-xl*. These models, without any fine-tuning, demonstrated higher cosine similarity for matched pairs compared to mismatched ones. This finding suggests that the ability to detect such equivalence is an inherent feature of high-quality sentence embedding models, not merely an *artifact* of fine-tuning. However, OpenAI *text-embedding-3-large* model is bad for its accessibility, a lightweight encoder *all-roberta-large-v1* allows for better speed and improved accuracy through fine-tuning, which is good in practice. The performance of the fine-tuned *roberta* model is shown in Figure 4. The substantial improvement in the model's capacity to identify hard negatives – mismatched pairs with subtle differences – is a direct result of our dedicated training weights allocation. We deliberately designed our training data selection to include a ratio of easy, semi-hard, and hard negatives as $[0.0, 0.4, 0.6]$,

[2]A reference domain model is only used for reference when we create manipulated versions of action schemas. We do this to recognize that natural language can be interpreted in various ways, rather than presupposing a one-to-one correspondence with a single ground truth schema, as discussed in Sec 1.

[3]International Planning Competition, a benchmark event for automated planning systems using PDDL.

[4]Business school students with no prior knowledge of PDDL programming or computational logic

Manipulation Type	Description	Example
Swap	Exchanges a predicate between preconditions and effects	Precondition: (at ?x ?y) Effect: (not (at ?x ?z)) $\rightarrow$ Precondition: (not (at ?x ?z)) Effect: (at ?x ?y)
Negation	Negates a predicate in either preconditions or effects	Precondition: (clear ?x) $\rightarrow$ Precondition: (not (clear ?x))
Removal	Removes a predicate from either preconditions or effects	Precondition: (and (on ?x ?y) (clear ?x)) $\rightarrow$ Precondition: (on ?x ?y)
Addition	Adds mutually exclusive (mutex) predicates to preconditions or effects (Helmert 2009)	Effect: (on-table ?x) $\rightarrow$ Effect: (and (on-table ?x) (holding ?x))

Table 1: Types of Manipulations for Generating Synthesized Hard Negative Action Schemas in Training Data. Mutexes are predicates that cannot be true simultaneously, e.g., one cannot hold a book and have it on a table simultaneously.

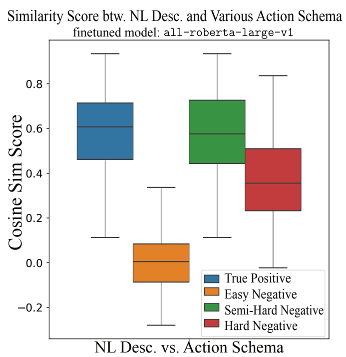

Figure 4: The sentence encoder enhances the identification of mismatched pairs by fine-tuning with negative samples.

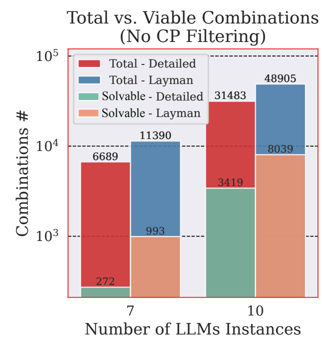

Figure 5: More LLM instances generate more solvable schemas without an expert.

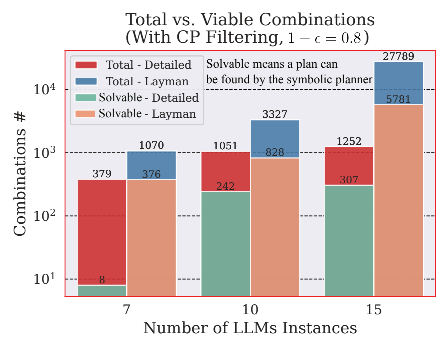

Figure 6: With CP, a large number of candidates are pruned, thereby improving efficiency.

respectively (see Appendix E.7). This ratio was strategically chosen to concentrate on hard negatives, as LLMs are more likely to make hard-negative mistakes when generating action schemas. By prioritizing hard negatives in our training dataset, we aimed to enhance the model's ability to filter out low-quality action schemas during the semantic coherence filtering step.

5.3 Pipeline Performance and Efficiency

Our pipeline's performance and efficiency are highlighted through several key observations. Firstly, the use of action schema library effectively produces *solvable* action schema sets without requiring expert-in-the-loop, as demonstrated in Figure 5. Notably, deploying 10 LLM instances is sufficient to generate solvable schema sets for all test domains, supporting **H3**. Secondly, Figures 5 and 6 reveal a clear pattern: when confronted with inherently ambiguous ***layman*** descriptions from non-expert participants, our pipeline generates a significantly increased number of distinct solvable schema sets (e.g., from 3419 to 8039 when LLM# = 10 w/o CP), thereby supporting **H2**. This increase is primarily attributed to the diverse selection of predicates within the action schemas. Each predicate selection reflects a different interpretation of the problem, with each schema set *emphasizing distinct features* deemed critical for planning.

Model	Planning Mechanism	Expert Input #	Action Schema #	Plan #	Heuristic Type	Soundness w.r.t. Schemas
Tree-of-Thought (Yao et al. 2024)	Pure LLM	0	N/A	Multiple	Self Verification	No
Guan et al. (2023)	Hybrid	≈ 59	Single	Single	Expert Validation	Yes
Ours	Hybrid	≤ 1	Multiple	Multiple	Semantic Sim. Scores	Yes

Table 2: Contrasts Our Pipeline with Existing Works. Note that the property of generating sound (logical correct) plans has been highlighted as a feature of the hybrid planner in prior work (Liu et al. 2023; Guan et al. 2023). However, there is no guarantee that the schemas are fully correct w.r.t. what the user actually wants. Thus, we are weakening the property to soundness w.r.t. schemas.

For instance, in the *Libraryworld* domain, we observed that some schema sets generated by some LLM instances

	Rank 1st	Rank 2nd	Rank 3rd	Rank 4th	Rank 5th	Avg. Rank
Gold	14	4	4	1	1	1.79
Ours	4	18	11	5	10	2.97
ToT	6	2	9	18	13	3.62

Table 3: Blind plan ranking eval.: Four assessors compared the top two plans from each approach to gold plans.

Model	Action Plan	Score
ToT GLM	Take book2 from table, Place book2 on book3, Take book1 from table, Place book1 on book2	Heuristic: 9.0
ToT GPT-3.5	Take book2, Place book2 on book3, Place book1 on table, Place book1 on book2, Remove book3, Place book3 on book2, Place book1 on book2, Check out book1	Heuristic: 5.11
ToT GPT-4o (1)	Take book2 from table, Place book2 on book3, Take book1 from table, Place book1 on book2	Heuristic: 8.5
ToT GPT-4o (2)	Take book2 from table, Take book3, Place book3 on table, Place book2 on table, Take book3 from table, Place book2 on table, Place book3 on book1, Take book2 from table, Place book2 on book3, **Depth limit reached**	Heuristic: 7.89
Ours GLM (1)	Remove book3 from book1, Take book1 from table, Place book1 on book2, Take book2 from table, Place book2 on book3	RankScr: 0.724
Ours GLM (2)	Remove book3 from book1, Take book2 from table, Place book2 on book3, Take book1 from table, Place book1 on book2	**RankScr: 0.788**

Table 4: Performance on Sussman Anomaly problem of ToT approach vs. ours. Both approaches generate multiple plans: ToT uses beam search, while ours generates multiple plans by feeding diverse sets of action schemas into a classical planner, with each set producing its own corresponding plan.

take into account the ‘category’ property of books when constructing actions such as stacking books on a shelf. This means that, according to these schema sets, only books within the same category can be stacked together, which is a more organized way of arranging books. Consequently, this leads to different planning outcomes that reflect the varied interpretations of the user query at hand, which are a direct result of the ambiguity present in the layman’s description and the flexibility it provides to LLMs in making such choices.

The pipeline’s ability to generate a range of potential interpretations in response to ambiguous inputs is a critical advantage. It ensures that all intended aspects of the user’s description can be captured, even when the description is imprecise or incomplete.

Thirdly, the integration of conformal prediction in the filtering step demonstrates a significant improvement in efficiency, as evidenced by Figure 6. With the confidence level $1 - \epsilon$ set to 0.8, the pipeline filtered out a large number of candidates, reducing the total number of combinations to 3.3% of the original (1051 out of 31483) but meanwhile, the ratio of solvable schemas (verified by the planner) increased from 10.9% to 23.0%. This result strongly supports **H3**, highlighting the pipeline’s ability to efficiently generate solvable and semantically coherent schema sets. See Table 2 for a comprehensive comparison of our pipeline with existing LLM-based planning approaches. Notably, the initial low ratio of solvable schema sets (10.9%) underscores the challenge faced within the LLM-symbolic planning paradigm, which may explain why expert intervention has been a common practice in the past.

5.4 Human Evaluation on Plan Quality

To further validate our approach, we conducted a human evaluation comparing the top two plan candidates generated by our pipeline against those from the ToT framework and a gold-standard plan derived from the reference PDDL domain model. Four expert assessors with extensive PDDL experience ranked the plans based on their feasibility in solving the given problems. The results, summarized in Table 3, clearly support **H4**.

For a deeper insight into our pipeline’s capabilities, we specifically tested the Sussman Anomaly, a well-known planning problem that requires simultaneous consideration of multiple subgoals, as solving them in the wrong order can undo previous progress (see Figure 1). As shown in Table 4, ToT approaches using various LLMs, including state-of-the-art models like GPT-4o, consistently fail to solve this problem. The failure arises from the mistaken assumption that the first subgoal mentioned (i.e., placing book 1 on top of book 2) should be addressed first, leading to incorrect plans. Interestingly, GPT-3.5 and GPT-4o exhibited different behaviors when faced with this problem. While GPT-3.5 consistently, yet incorrectly, asserted it had completed the problem, GPT-4o occasionally exhibited awareness of the plan’s incompleteness. However, even with this heightened awareness, GPT-4o was unable to identify the correct path within the given depth limit. In contrast, our pipeline generates a range of plans, including suboptimal ones, but excels at identifying and prioritizing the most promising candidates through its ranking process that is based on the cumulative cosine similarity scores of generated action schemas. By strictly adhering to semantic alignment between these schemas and natural language descriptions, and by using a symbolic planner, the system avoids being misled by the tendency – observed in both humans and LLMs – to reason in a linear manner. This tendency involves prioritizing subgoals based on their order of appearance rather than considering their underlying logical dependencies. Such linear reasoning can lead to noninterleaved planning, where subgoals are tackled in the order they are presented and each must be fully completed before addressing the next one, which is a pitfall in complex planning problems like the Sussman Anomaly.

5.5 Failure Case Analysis

Schema Set with No Plan Found: We encountered instances where no solvable action schema set was generated, primarily due to limitations in the LLM’s reasoning capabilities. The use of open-source LLMs, while more accessible, may result in a lower success rate compared to more advanced proprietary models like GPT-4o. Specifically, with 7 LLM instances, we observed occasional failures of generating solvable sets action schemas for the *libraryworld* and *minecraft* domains. Nevertheless, solvable schema sets were consistently obtained across all domains when the number of LLM instances was increased to 10 (see Appendix F for a breakdown of schema set yield by LLM instance count).

Unexpected Preference: In the *Dungeon* domain, human assessors unexpectedly preferred ToT-generated plans over both the reference plan and the proposed pipeline's plans. Further analysis revealed that the ToT plans consistently included a step: *grabbing a sword*. Interestingly, grabbing a sword was not a necessary step for solving the given problem. Consequently, symbolic planners, focused on optimal pathfinding, excluded this step from their plans. However, this "unnecessary" step of acquiring a sword aligns with common strategies in Dungeon games, where players typically prioritize preparedness. Thus, this action strongly appealed to human assessors, causing them to rank the ToT-generated plans higher.

6 Conclusion

Existing LLM-symbolic planners offer limited and potentially biased schema options due to expert-in-the-loop. Our work presents a 3-step pipeline that learn symbolic PDDL models over ambiguous natural language descriptions. Our findings demonstrate that a full end to end hybrid planner is possible without expert intervention, paving the way for democratizing planning systems for a broader audience. One limitation in this work is the lack of direct evaluation methods for assessing the quality of generated action schema sets. Metrics like "bisimulation" (Coulter et al. 2022) or "heuristic domain equivalence" (Oswald et al. 2024) require the generated schema sets to have the same action parameters as a predefined reference set. This approach doesn't suit our context, where action parameters are flexible and inferred in real-time from natural language descriptions.

Acknowledgements

Sukai Huang is supported by Melbourne Research Scholarship established by The University of Melbourne.

This research was supported by The University of Melbourne's Research Computing Services and the Petascale Campus Initiative.

References

Aeronautiques, C.; Howe, A.; Knoblock, C.; McDermott, I. D.; Ram, A.; Veloso, M.; Weld, D.; Sri, D. W.; Barrett, A.; Christianson, D.; et al. 1998. Pddl— the planning domain definition language. *Technical Report, Tech. Rep.*

Ahn, M.; Brohan, A.; Brown, N.; Chebotar, Y.; Cortes, O.; David, B.; Finn, C.; Gopalakrishnan, K.; Hausman, K.; Herzog, A.; Ho, D.; Hsu, J.; Ibarz, J.; Ichter, B.; Irpan, A.; Jang, E.; Ruano, R. M. J.; Jeffrey, K.; Jesmonth, S.; Joshi, N. J.; Julian, R. C.; Kalashnikov, D.; Kuang, Y.; Lee, K.-H.; Levine, S.; Lu, Y.; Luu, L.; Parada, C.; Pastor, P.; Quiambao, J.; Rao, K.; Rettinghouse, J.; Reyes, D. M.; Sermanet, P.; Sievers, N.; Tan, C.; Toshev, A.; Vanhoucke, V.; Xia, F.; Xiao, T.; Xu, P.; Xu, S.; and Yan, M. 2022. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *Conference on Robot Learning*.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*, 129(1-2): 5–33.

Chen, M.; Tworek, J.; et al. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374.

Chrpa, L.; et al. 2017. The fifth international competition on knowledge engineering for planning and scheduling: Summary and trends. *AI Magazine*, 38(1): 104–106.

Coulter, A.; Ilie, T.; Tibando, R.; and Muise, C. 2022. Theory Alignment via a Classical Encoding of Regular Bisimulation. In *Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling (KEPS) at ICAPS*. ICAPS.

Guan, L.; Valmeekam, K.; Sreedharan, S.; and Kambhampati, S. 2023. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36: 79081–79094.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; Muise, C.; Brachman, R.; Rossi, F.; and Stone, P. 2019. *An introduction to the planning domain definition language*, volume 13. Springer.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5-6): 503–535.

Hou, Y.; Zhang, J.; Lin, Z.; Lu, H.; Xie, R.; McAuley, J.; and Zhao, W. X. 2023. Large Language Models are Zero-Shot Rankers for Recommender Systems. In *European Conference on Information Retrieval*.

Hou, Z.; Niu, Y.; Du, Z.; Zhang, X.; Liu, X.; Zeng, A.; Zheng, Q.; Huang, M.; Wang, H.; Tang, J.; and Dong, Y. 2024. ChatGLM-RLHF: Practices of Aligning Large Language Models with Human Feedback. arXiv:2404.00934.

Huang, L.; Yu, W.; Ma, W.; Zhong, W.; Feng, Z.; Wang, H.; Chen, Q.; Peng, W.; Feng, X.; Qin, B.; and Liu, T. 2024. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. *ACM Transactions on Information Systems*.

Huang, W.; Abbeel, P.; Pathak, D.; and Mordatch, I. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, 9118–9147. PMLR.

Kahneman, D. 2011. *Thinking, fast and slow*. macmillan.

Kambhampati, S.; Valmeekam, K.; Guan, L.; Verma, M.; Stechly, K.; Bhambri, S.; Saldyt, L. P.; and Murthy, A. B. 2024. Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. In *Forty-first International Conference on Machine Learning*.

Katz, M.; et al. 2024. Thought of Search: Planning with Language Models Through The Lens of Efficiency. arXiv:2404.11833.

Kojima, T.; Gu, S. S.; Reid, M.; Matsuo, Y.; and Iwasawa, Y. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213.

Lin, S. C.; et al. 2022. Teaching Models to Express Their Uncertainty in Words. *Trans. Mach. Learn. Res.*, 2022.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. In *AAAI Conference on Artificial Intelligence*.

Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; Biswas, J.; and Stone, P. 2023. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. arXiv:2304.11477.

Moravcsik, J. M. 1983. Natural languages and formal languages: a tenable dualism. In *Language, Logic and Method*, 225–239. Springer.

Oswald, J.; Srinivas, K.; Kokel, H.; Lee, J.; Katz, M.; and Sohrabi, S. 2024. Large Language Models as Planning Domain Generators. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 423–431.

Pallagani, V.; Roy, K.; Muppasani, B.; Fabiano, F.; Loreggia, A.; Murugesan, K.; Srivastava, B.; Rossi, F.; Horesh, L.; and Sheth, A. 2024. On the Prospects of Incorporating Large Language Models (LLMs) in Automated Planning and Scheduling (APS). In *International Conference on Automated Planning and Scheduling*.

Robinson, J. D.; Chuang, C.-Y.; Sra, S.; and Jegelka, S. 2023. Contrastive Learning with Hard Negative Samples. In *International Conference on Learning Representations*.

Rozière, B.; Gehring, J.; et al. 2024. Code Llama: Open Foundation Models for Code. arXiv:2308.12950.

Sadinle, M.; et al. 2019. Least ambiguous set-valued classifiers with bounded error levels. *Journal of the American Statistical Association*, 114(525): 223–234.

Silver, T.; and Chitnis, R. 2020. PDDLGym: Gym Environments from PDDL Problems. In *ICAPS Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*.

Silver, T.; Dan, S.; Srinivas, K.; Tenenbaum, J. B.; Kaelbling, L.; and Katz, M. 2024. Generalized planning in pddl domains with pretrained large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 20256–20264.

Sloman, S. A. 1996. The empirical case for two systems of reasoning. *Psychological Bulletin*, 119: 3–22.

Valmeekam, K.; Marquez, M.; Sreedharan, S.; and Kambhampati, S. 2023. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36: 75993–76005.

Valmeekam, K.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2022. PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change. In *Neural Information Processing Systems*.

Wang, Z.; Cai, S.; Chen, G.; Liu, A.; Ma, X.; and Liang, Y. 2023. Describe, Explain, Plan and Select: Interactive Planning with LLMs Enables Open-World Multi-Task Agents. In *Neural Information Processing Systems*.

Weaver, W. 1952. Translation. In *Proceedings of the Conference on Mechanical Translation*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.

Xiang, J.; Tao, T.; Gu, Y.; Shu, T.; Wang, Z.; Yang, Z.; and Hu, Z. 2024. Language models meet world models: Embodied experiences enhance language models. *Advances in neural information processing systems*, 36.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; and Narasimhan, K. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Zhuang, S.; Liu, B.; Koopman, B.; and Zuccon, G. 2023. Open-source Large Language Models are Strong Zero-shot Query Likelihood Models for Document Ranking. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 8807–8817.

A Probabilistic Analysis for Action Schema Combinations

Let's consider a domain consisting of M actions, where the probability of generating a correct action schema for each action is p. The probability of obtaining a solvable (i.e., correctly formed and usable by the symbolic planner) set of action schemas for the domain is then p^M. p^M diminishes exponentially as M increases. For instance, with $p = 0.05$ (based on observations from Guan et al. (2023)) and $M = 5$, the probability of obtaining a solvable set of action schemas from vanilla LLMs is only 0.00003125%. In contrast, by combining the action schemas generated by each LLM instance, we can obtain approximately $\binom{N}{1}^M$ different sets of action schemas, resulting in a vast number of action schema set candidates.

Success Analysis: Assume each action schema α_{ij} has an independent probability p_{ij} of being solvable. For simplicity, let's assume $p_{ij} = p$ for all i and j. A set of action schemas in the PDDL domain model is considered as solvable if all its action schemas are solvable. Therefore, the probability that a generated set of action schemas is not solvable is $(1 - p^M)$, and the probability that none of the generated sets are solvable is $(1 - p^M)^{N^M}$ and the probability of at least one combination is solvable is $1 - (1 - p^M)^{N^M}$. For large N, as long as $p \in (0, 1)$, $(1 - p^M)^{N^M}$ approaches 0. For example, with 10 LLM instances, a domain of 5 actions, and a solvability probability of 0.05, the probability of obtaining at least one solvable set of action schemas is $1 - (1 - (0.05)^5)^{5^{10}} = 1 - 0.0484 = 95.2\%$.

B Conformal Prediction Details

With a user-specified confidence level $1 - \epsilon$, we calculate the $\frac{\lceil (n+1)(1-\epsilon) \rceil}{n}$ empirical quantile of the cosine similarity scores for true positive pairs in the calibration set, denoted as $\hat{q}$, where n is the size of the calibration data pairs. Statistically, it ensures that at least $1 - \epsilon$ fraction of the true positive pairs will have a cosine similarity score greater than $\hat{q}$. We then use $\hat{q}$ as the threshold to filter out action schemas with cosine similarity scores below this value. Statistically, this approach guarantees a high probability of preserving true positive action schema candidates while minimizing the size of the filtered set (Sadinle et al. 2019). For an illustration of this process, see *step 2* in Figure 3. The pseudo-code for calculating the empirical quantile is shown in Algorithm 1.

During our experiments, we set $\epsilon = 0.2$ to ensure a high confidence level while maintaining a reasonable number of solvable action schema candidates. This choice was based on the trade-off between the confidence level and the number of solvable action schema candidates generated. A higher ϵ leads to a more stringent filtering process, resulting in a smaller number of solvable action schema candidates so as to save computational resources. In our experiments, the pretrained model *all-roberta-large-v1*'s $\hat{q}$ value at $\epsilon = 0.2$ was 0.398. Thus, any action schema with a cosine similarity score below 0.398 was filtered out, ensuring that only high-quality action schemas were passed to the symbolic planner.

Algorithm 1: Calculate Empirical Quantile ($\hat{q}$)

Require: Calibration set $\{\mathcal{Z}(\alpha), \alpha\}_n$, where $\mathcal{Z}(\alpha)$ is natural language description of the corresponding true action schema α;
n is the size of the calibration set;
Significance level ϵ;
Sentence encoder model E

1: ▷ *Obtain the Cosine Similarity Score* ◁
2: $\{e\}_n \leftarrow \left\{ \frac{E(\mathcal{Z}(\alpha)) \cdot E(\alpha)}{\|E(\mathcal{Z}(\alpha))\| \|E(\alpha)\|} \right\}_n$
3: ▷ *Compute quantile level* ◁
4: $q_{\text{level}} \leftarrow \frac{\lceil (n-1) \times (1-\epsilon) \rceil}{n}$
5: ▷ *Compute empirical quantile* ◁
6: $\hat{q} \leftarrow$ *np.quantile*($\{e\}_n$, q_{level}, *method='lower'*)
7: **return** $\hat{q}$

C Domain Leakage Investigation Methodology

To determine whether the testing PDDL domains had been leaked to the training data of the large language models (LLMs), we conducted a specific investigation. This involved providing the LLMs with partial information about the PDDL domains, specifically the types, predicates, and the first action schema. Following this, we asked the LLMs to generate the remaining content of the domain.

In our findings, the LLMs were able to accurately generate the complete action schemas for the well-known *blocksworld*, *tyreworld*, and *logistics* environments, which have been extensively used in prior research, indicating that LLMs had likely been exposed to these domains during pretraining. In contrast, for the testing domains used in our experiments, namely *minecraft*, *dungeon*, and *libraryworld*, the LLMs were unable to reconstruct the complete action schemas based on the partial information provided, thereby ensuring the integrity of our experimental conditions.

D Symbolic Planner Details

Modern symbolic planners can efficiently verify the solvability of a generated schema set by leveraging its ability to prove delete-free reachability in polynomial time (Bonet and Geffner 2001). This leads to very quick verification of whether the generated schema set is viable for generating a plan. Additionally, modern planners can run efficiently on a single CPU thread, and multiple problem instances can be solved in parallel if more CPUs are available. For instance, an AMD Ryzen 5900 with 32 threads can easily check the solvability of up to 20,000 generated schema sets within 2 minutes. By applying a Conformal Prediction (CP) filtering mechanism, we can further reduce the number of schema sets that need to be checked, allowing us to verify solvability within seconds, as most unsuitable sets are filtered out.

E Experiments Details

This section consists of the following:

- §E.1: Detailed natural language descriptions and reference PDDL models for the testing domains *Libraryworld* and *Dungeon*.
- §E.2: Details on the CO-STAR prompt engineering framework used for LLM prompt engineering.
- §E.3: Prompt template used for generating action schemas, including how to structure few-shot learning examples in the prompt and how to obtain CoT reasoning examples automatically from advanced LLMs.
- §E.4: Prompt template used for Tree-of-Thought direct LLM-based planning.
- §E.5: Syntax correction process for the generated action schemas.
- §E.7: LLM model configurations and training configurations for the sentence encoder model.
- §F.1: Additional experiments and results, that are outside of the scope of the hypotheses but provide additional insights into the pipeline's performance.

E.1 Testing Tasks Specifications

```
Domain Description: This domain is structured to allow organizing and managing books within a
     library setting. The actions and predicates support the movement of books between tables
     and shelves, ensuring that conditions like accessibility and the librarian's hands being
     free are met. Additionally, it includes managing book categories, shelf space, and check
    -out/return processes to reflect a more complex library system.

Action Description:
take-from-table:
    detailed: Imagine you're a librarian managing a table full of books. The 'take-from-table
    ' action allows you to pick up a book that is on the table, provided it is accessible and
     your hands are free. This action simulates the scenario where you find a book on the
    table, ensure it's not covered by any other book, and then pick it up, thus holding it in
     your hands.
    ambiguous: Pick up a book from the table if it's not covered and your hands are empty.
place-on-shelf:
    detailed: Consider a librarian holding a book and standing near a shelf. The 'place-on-
    shelf' action involves placing the held book on top of another book on the shelf, given
    that the book on the shelf is accessible. This action results in the held book becoming
    accessible, the book on the shelf becoming inaccessible, and the librarian's hands
    becoming free.
    ambiguous: Put a book you're holding on top of another accessible book on the shelf.
...
Predicate List:
(on-shelf ?x ?y - book) ;; ?x is on top of ?y on the shelf
(on-table ?x - book) ;; ?x is on the table
(accessible ?x - book) ;; ?x is accessible (not covered)
(hands-free) ;; The hands of the librarian are free
(holding ?x - book) ;; The librarian is holding ?x
(belongs-to-category ?x - book ?cat - category) ;; ?x belongs to the category ?cat
(shelf-empty ?cat - category) ;; The shelf for category ?cat is empty
(shelf-overflow ?cat - category) ;; The shelf for category ?cat is full
(book-request ?book - book) ;; There is a request for book ?book
(return-due ?book - book) ;; Book ?book is due for return
(checked-out ?book - book) ;; Book ?book is checked out
```

Listing 1: Libraryworld Domain Descriptions

```
Initial State Description: In the library, there are three books: Book1, Book2, and Book3.
    Book3 is on top of Book1 and they are both on the shelf, while Book2 is on the table.
    Book1 can also be considered as on the table it is just at the bottom of the shelf. Both
    Book2 and Book3 are accessible, meaning they can be interacted with. The library worker's
     hands are free. Book1 belongs to the Fiction category, Book2 belongs to the NonFiction
    category, and Book3 belongs to the Reference category.

Goal State Description: The goal is to have Book2 on top of Book3, and also Book1 on top of
    Book2.
```

Listing 2: Libraryworld Problem Descriptions

```
(define (domain libraryworld)
(:requirements :strips :typing :negative-
   preconditions)
(:types book category)
(:predicates
    (on-shelf ?x ?y - book)
    (on-table ?x - book)
    (accessible ?x - book)
    (hands-free)
    ...
)
(:action take-from-table
    :parameters (?x - book)
    :precondition (and (accessible ?x) (on
   -table ?x) (hands-free))
    :effect (and (not (on-table ?x))
                 (not (accessible ?x))
                 (not (hands-free))
                 (holding ?x))
)
(:action place-on-table
    :parameters (?x - book)
    :precondition (holding ?x)
    :effect (and (not (holding ?x))
                 (accessible ?x)
                 (hands-free)
                 (on-table ?x))
)
...
)
```

Listing 3: Libraryworld Reference PDDL Domain Model

```
(define (problem organize-books)
(:domain libraryworld)
(:objects
    Book1 Book2 Book3 - book
    Fiction Non_Fiction Reference -
    category
)
(:init
    (on-table Book1)
    (on-shelf Book3 Book1)
    (on-table Book2)
    (accessible Book2)
    (accessible Book3)
    (hands-free)
    (belongs-to-category Book1 Fiction)
    (belongs-to-category Book2 Non_Fiction
    )
    (belongs-to-category Book3 Reference)
    ...
)
(:goal
    (and
        (on-shelf Book2 Book3)
        (on-shelf Book1 Book2)
    )
)
)
```

Listing 4: Libraryworld Reference PDDL Problem Model

```
Domain Description: Help the hero to get out of dungeon! A hero woke up in a dungeon full of
   monsters and traps (perhaps the party last night went wrong...) and needs your help to
   get out.  Here are basic facts for the dungeon domain: - The dungeon contains rooms that
   are **connected** by corridors (dungeon can thus be represented by undirected graph) -
   each room can be **empty**, or can have a **monster** in it, or can have a **trap** in it
   , or can have a **sword** in it - one of the empty rooms is the **goal**: it has an exit,
    so the hero can escape.

Action Description:
move:
    detailed: The hero can **move** to an adjacent room (connected by a corridor) that has
   not been destroyed (i.e., the hero has not already visited the room). When this action is
    executed, the original cell get destroyed.
    ambiguous: Hero can move if the - hero is at current location - cells are connected, -
   there is no trap in current loc, and - destination does not have a trap/monster.
pick-sword:
    detailed: **Pickup** the sword if present in the room the hero is currently in and the
   hero is empty handed.
    ambiguous: Hero picks a sword if he's in the same location.
...
Predicate List:
(at-hero ?loc - cells) ;;  Hero's cell location
(at-sword ?s - swords ?loc - cells) ;; Sword cell location
(has-monster ?loc - cells) ;; Indicates if a cell location has a monster
(has-trap ?loc - cells) ;; Indicates if a cell location has a trap
(is-destroyed ?obj) ;; Indicates if a chell or sword has been destroyed
(connected ?from ?to - cells) ;; connects cells
(arm-free) ;; Hero's hand is free
(holding ?s - swords) ;; Hero's holding a sword
(trap-disarmed ?loc) ;; It becomes true when a trap is disarmed
```

Listing 5: Dungeon Domain Descriptions

```
Initial State Description: In the dungeon, the hero starts at cell5, with free hands ready
    for action. The hero is aware of the dungeon's layout, which consists of multiple
    interconnected cells. A sword is located in cell4. The dungeon contains dangerous
    monsters located in cell3 and cell8, and a trap is present in cell2. The hero must
    navigate this treacherous environment, using the connections between the cells to move
    around. The connections are as follows:

    Cell1 is connected to cell2.
    Cell2 is connected to cell1 and cell3.
    Cell3 is connected to cell2 and cell4.
    Cell4 is connected to cell3 and cell5.
    Cell5 is connected to cell4 and cell8.
    Cell6 is connected to cell7.
    Cell7 is connected to cell6 and cell8.
    Cell8 is connected to cell7 and cell5.
    Cell2 is also connected to cell6.
    Cell3 is also connected to cell7.
    Cell4 is also connected to cell8.

Goal State Description: The hero's ultimate objective is to reach cell1 safely.
```

Listing 6: Dungeon Problem Descriptions

```
(define (domain rpggame)
(:requirements :typing :negative-
    preconditions
)
(:types
    swords cells
)
(:predicates
    (at-hero ?loc - cells)
    (at-sword ?s - swords ?loc - cells)
    (has-monster ?loc - cells)
    (has-trap ?loc - cells)
    ...
)
(:action move
    :parameters (?from ?to - cells)
    :precondition (and
        (connected ?from ?to)
        (at-hero ?from)

        (not (has-trap ?from))
        (not (is-destroyed ?to))
        (not (has-trap ?to))
        (not (has-monster ?to))

    )
    :effect (and
        (at-hero ?to)
        (is-destroyed ?from)
        (not (at-hero ?from))
    )
)

...
)
```

Listing 7: Dungeon Reference PDDL Domain Model

```
(define (problem p1-dangeon)
(:domain rpggame)
(:objects
    cell1 cell2 cell3 cell4 cell5 cell6
   cell7 cell8 - cells
    sword1 - swords
)
(:init

    ;Initial Hero Location
    (at-hero cell5)
    ;He starts with a free arm
    (arm-free)
    ;Initial location of the swords
    (at-sword sword1 cell4)
    ;Initial location of Monsters
    (has-monster cell3)
    (has-monster cell8)
    ;Initial location of Traps
    (has-trap cell2)

    ;Graph Connectivity
    (connected cell1 cell2)
    (connected cell2 cell1)
    (connected cell2 cell3)
    (connected cell3 cell2)
    (connected cell3 cell4)
    (connected cell4 cell3)
    (connected cell4 cell5)
    ...
)
(:goal (and
            (at-hero cell1)
))

)
```

Listing 8: Dungeon Reference PDDL Problem Model

E.2 CO-STAR Framework

The CO-STAR framework is a structured template for crafting effective prompts for LLMs. Developed by GovTech Singapore's Data Science and Artificial Intelligence Division, CO-STAR is designed to improve the quality of LLM-generated responses by systematically addressing key aspects that influence output.

The CO-STAR acronym stands for: **Context (C)**: Provide background information on the task; **Objective (O)**: Define the specific task you want the LLM to perform; **Style (S)**: Specify the desired writing style for the LLM's response; **Tone (T)**: Set the attitude or emotional quality of the response; **Audience (A)**: Identify the intended recipients of the response; **Response (R)**: Outline the expected format of the response.

E.3 Prompt Template for Action Schema Generation and Obtaining CoT Reasoning Examples

Based on the CO-STAR framework, we designed a structured prompt template for generating action schemas. The template includes the following components:

````
System: # CONTEXT #
You are a tool called PDDL Modeling
    Assistant. \
You are a technical experts in constructing
    Planning Domain Definition Language (
    PDDL) models via the natural language
    context.

# OBJECTIVE #
* Construct parameters, preconditions and
    effects based on the domain information,
     action description and the action name.
* All variables in the preconditions and
    effects must be listed in the action's
    parameters. This restriction helps
    maintain the action's scope and prevents
     ambiguity in the planning process.
* Do not use predicates that are not defined
     in the available predicates list to
    construct the preconditions and effects.
* When the natural language description is
    ambiguous or certain predicate changes
    are implied, make reasonable assumptions
     based on common sense to fill up the
    implicit predicate in the PDDL action.

# STYLE #
Follow the writing style of technical
    experts. The output can be parsed by a
    machine, so it is important to follow
    the structured format.

# TONE #
Be precise and concise in constructing the
    PDDL action. The PDDL action should be
    clear and unambiguous.

# AUDIENCE #
Your audience is someone who is trying to
    learn how to construct PDDL actions from
     natural language descriptions.

# RESPONSE #
The response should be in the following
    format:
---
**Explanation:** [Your explanation here]

**Response:**
Parameters:
1. ?x - [type]: [parameter description]
2. ...

Preconditions:
```
(and
    ([predicate_1] ?x)
)
```

Effects:
```
(and
    (not ([predicate_2] ?x))
    ([predicate_2] ?x)
    ...
)
```
---
````

We further include two example query-answer pairs to facilitate in-context learning for LLMs. The example are taken from the training domains rather than the testing domains to ensure the integrity of the experimental conditions.

```
Example Human: One or two examples from
    other domains for illustrating the input
     and output formats.
Here are two examples from the newspapers
    domain for demonstrating the output
    format.

Domain information: This domain is
    structured to model a simple newspaper
    delivery system, where the main tasks
    involve picking up newspapers from a
    home base, moving to different locations
    , and delivering the papers to locations
     that require them.

A list of available predicates
1. (at ?loc - loc) ;; the agent is at loc
    location
2. (is_Home_Base ?loc - loc) ;; the location
     is the home base
3. (satisfied ?loc - loc) ;; the location
    has received the newspaper
4. (wants_Paper ?loc - loc) ;; the location
    needs a newspaper
5. (unpacked ?paper - paper) ;; the paper is
     unpacked
6. (carrying ?paper - paper) ;; the agent is
     carrying the paper
Example 1
```

```
Action Description: RoboDelivery is at the
    home base (loc_home).
There is an unpacked newspaper (paper1) at
    the home base.
RoboDelivery performs the pick-up action.

    Preconditions: RoboDelivery is at
    loc_home (which is the home base), and
    paper1 is unpacked.
    Effects: RoboDelivery is now carrying
    paper1, and paper1 is no longer unpacked
    .

Action name: pick-up


Your answer:
---

Example AI: **Explanation:**

Let's imagine a situation where someone,
    like a delivery person, needs to pick up
     newspapers from a place, like a home
    base, and deliver them to various
    locations. This situation can be broken
    down into smaller, specific steps to
    make it easy for a computer to
    understand and execute. This is where
    the action schema you mentioned comes
    into play.

The action is called "pick-up," and it
    involves the delivery person picking up
    newspapers to deliver. This action is
    defined in a language called PDDL, which
     is like giving a computer a recipe to
    follow.

1. **Parameters**: These are like
    ingredients in a recipe. Here, the
    ingredients are any newspaper (`?paper`)
     and a location (`?loc`), which in this
    scenario is the home base.

2. **Preconditions**: These are conditions
    that must be true before the action can
    start. Think of it like checking if you
    have all your ingredients before you
    start cooking.
   - `(at ?loc)`: The delivery person must
    be at the location specified (`?loc`).
    This makes sure they are actually at the
     home base to pick up the newspapers.
   - `(is_Home_Base ?loc)`: The location
    where they are must be the home base.
    This is crucial because newspapers ready
     to be delivered are stored here.
   - `(unpacked ?paper)`: The newspapers
    need to be unpacked and ready to be
    picked up. It ensures the newspapers are
     not in boxes or still bundled up,
    making them easy to grab and go.

3. **Effects**: These describe what happens
    after the action is completed.
   - `(not (unpacked ?paper))`: The
    newspaper is no longer in an unpacked
    state at the base because it has been
    picked up.
   - `(carrying ?paper)`: The newspaper is
    now being carried by the delivery person
    , ready to be delivered to someone who
    needs it.

This schema helps the computer understand
    exactly what needs to happen when the "
    pick-up" action is called in the program
    . By specifying preconditions, we ensure
     the action makes sense (you can't pick
    up a newspaper if you're not there or if
     the papers aren't ready). The effects
    tell the computer how the world changes
    after the action, like the newspaper
    moving from being unpacked at the base
    to being carried for delivery.

By breaking down tasks this way, we help the
     computer efficiently and correctly
    handle tasks that seem simple to us but
    are quite detailed when programmed.

**Response:**
Parameters:
1. ?from - loc: [short description of the
    parameter]
2. ?to - loc: [short description of the
    parameter]

Preconditions:
'''
(at ?from)
'''

Effects:
'''
(and
    (not (at ?from))
    (at ?to)
)
'''

---
```

As demonstrated in the example above, we include a detailed explanation section. This approach, known as Chain-of-Thought (CoT), has been shown to enhance LLMs' reasoning capabilities by explicitly prompting them to show intermediate steps in their reasoning process.

The following section presents the actual query-response pairs generated by the LLMs during the action schema generation process.

```
System: [SAME AS THE PREVIOUS EXAMPLE]
Human: Question: Here is the task.
A natural language description of the domain
Domain information: This domain is
    structured to allow organizing and
    managing books within a library setting.
```

```
    The actions and predicates support the
   movement of books between tables and
   shelves, ensuring that conditions like
   accessibility and the librarian's hands
   being free are met. Additionally, it
   includes managing book categories, shelf
    space, and check-out/return processes
   to reflect a more complex library system
   .

A list of available predicates
1. (on-shelf ?x ?y - book) ;; ?x is on top
   of ?y on the shelf
2. (on-table ?x - book) ;; ?x is on the
   table
3. (accessible ?x - book) ;; ?x is
   accessible (not covered)
4. (hands-free) ;; The hands of the
   librarian are free
5. (holding ?x - book) ;; The librarian is
   holding ?x
6. (belongs-to-category ?x - book ?cat -
   category) ;; ?x belongs to the category
    ?cat
7. (shelf-empty ?cat - category) ;; The
   shelf for category ?cat is empty
8. (shelf-overflow ?cat - category) ;; The
   shelf for category ?cat is full
9. (book-request ?book - book) ;; There is a
    request for book ?book
10. (return-due ?book - book) ;; Book ?book
   is due for return
11. (checked-out ?book - book) ;; Book ?book
    is checked out

Action Description: Imagine you're a
   librarian managing a table full of books
   . The 'take-from-table' action allows
   you to pick up a book that is on the
   table, provided it is accessible and
   your hands are free. This action
   simulates the scenario where you find a
   book on the table, ensure it's not
   covered by any other book, and then pick
    it up, thus holding it in your hands.

Action name: take-from-table


Your answer:

AI: **Explanation:**
In the context of a library, the "take-from-
   table" action represents the physical
   act of a librarian picking up a book
   from a table. This action is only
   feasible under certain conditions, which
    are represented by the preconditions of
    the action. The effects describe the
   outcome or the change in the state of
   the world once the action is executed.

The parameters of the action are the book
   that is to be picked up and potentially
   the table it is on, though the table is
   not explicitly mentioned in the
   available predicates, so we'll focus on
   the book.

The preconditions ensure that:
1. The book is on the table and accessible,
   meaning it's not obscured or covered by
   other books.
2. The librarian's hands are free,
   indicating that they are not currently
   holding any other book.

Once these conditions are met, the librarian
    can perform the "take-from-table"
   action, which results in two main
   effects:
1. The book is no longer on the table, as it
    is now being held by the librarian.
2. The librarian is no longer hands-free, as
    they are now holding the book.

This action is a fundamental part of
   managing books within the library,
   allowing the librarian to move books
   from tables to shelves or to handle
   checkouts and returns.

**Response:**
Parameters:
1. ?book - book: [The book that the
   librarian will pick up from the table]

Preconditions:
'''
(and
    (on-table ?book)
    (accessible ?book)
    (hands-free)
)
'''

Effects:
'''
(and
    (not (on-table ?book))
    (not (hands-free))
    (holding ?book)
)
'''
```

We can see that the LLM-generated response closely aligns with the expected format, demonstrating the effectiveness of the prompt template in guiding the LLM to generate accurate action schemas.

E.4 Prompt Template for ToT Direct Planning

The Tree-of-Thought (ToT) direct planning approach uses a prompt template similar in structure to that used for action schema generation. The key distinction lies in the system prompt, which instructs the LLM to generate plans rather than action schemas. The system prompt for ToT direct planning is as follows:

```
System: # CONTEXT #
```

```
You are a tool called Automated Planning
    Action Planner.
You are a technical expert in constructing
    and evaluating the quality of action
    plans via the natural language context.
# OBJECTIVE #
* Add only one more action to the given
    action plan sequence towards achieving
    the goal state.
* Provide the action name and the objects
    that this action will interact with.
* Evaluate the confidence score that
    continuing with the suggested action
    plan will eventually lead to the goal
    state.
* Determine if the action plan has already
    reached the goal state.
* Output only one action at each step and
    stop.
* You should think step by step, you think
    think more steps.
...
# RESPONSE #
The response should be in the following
    format:
---
**Response:**
1. Action {n}: [Action Name]
2. Objects:
   - ?x - [type]: [object description]
   - ...

3. Updated Action Plan:
   - [Action 1 and the objects involved]
   - [Action 2 and the objects involved]
   - ...
   - [Action {n} and the objects involved]

**Confidence Evaluation:**
[Analyze the updated action plan, then at
    the last line conclude "The confidence
    score is {s}", where s is an integer
    from 1 to 10]

**Goal State Check:**
[Briefly analyze the current state, then at
    the last line conclude "The planning is
    continuing" or "The planning is
    completed"]
---
```

E.5 Syntax Correction Process

After obtaining the raw outputs from the LLMs, our pipeline post-processes these outputs into structured PDDL snippets, correcting any potential syntax errors present in the initial generation. Although previous research has demonstrated that LLMs can effectively correct syntax errors when given feedback from validation tools (Guan et al. 2023; Rozière, Gehring et al. 2024; Silver et al. 2024), this approach is computationally expensive because it requires numerous iterative calls to the LLM.

As our work primarily focuses on addressing semantic (factual) errors in LLM outputs, we opted for a more efficient approach to syntax correction. Instead of using LLMs, we employed the PDDL parser tool from https://github.com/jan-dolejsi/vscode-pddl to directly correct syntax errors. This tool specifically addresses syntax errors, thus it will not introduce any semantic changes to the generated PDDL snippets.

E.6 Problem Instance Generation Process

Our work assume that problem instances are described with sufficient precision, allowing for a straightforward translation into PDDL problem snippets for use by an external symbolic planner. By "trivial," we mean that this translation can be effectively handled by a modern machine translation system. The core challenge of our study lies in interpreting the more ambiguous natural language descriptions of the planning task domain to generate varied solvable schema sets. Consequently, we assume an oracle problem instance translator in our pipeline, which can be easily implemented using existing tools like the PDDL parser mentioned in the syntax correction process.

E.7 Model and Training Configurations

LLM configuration is as follows:

Table 5: LLM model configuration

Parameter	Action Schema Generation	Direct Plan Generation
model_name	glm-4-0520	glm-4-0520, gpt-4o-2024-05-13, gpt-3-turbo
top_p	0.3	0.80
temperature	0.3	0.99
max_tokens	1024	1024
tree_breadth	-	3
tree_depth_ratio	-	1.5 tree depth = ratio * referece plan length)

The finetuning configuration for the sentence encoder model is as follows:

Table 6: Sentence Encoder Finetuning Configs

Parameter	Value
train_negative_weights	[0.0, 0.4, 0.6] (ratio of selecting easy, semi-hard, hard negatives from the training dataset)
train_batch_size	256
training_epoch	40
sentence_encoder	all-roberta-large-v1

Finetuning Details: The sentence encoder model was fine-tuned on a dataset of 200,000 synthesized samples generated by action schema manipulation. The process utilized an NVIDIA A100 80GB PCIe GPU on a Linux 5.14.0 system. It takes about 11 hours to complete the training process. More details can be found in the code repository.

F Extra Results

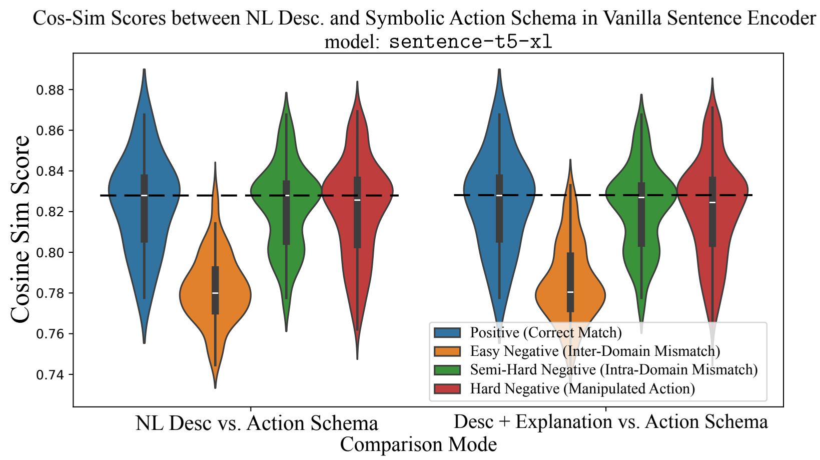

Figure 7: The pre-trained `sentence-t5-xl` sentence encoder demonstrates semantic alignment for matched action schemas and misalignment for mismatched ones, supporting the concept of semantic equivalence across different representations of the same entity.

Table 7: Detailed action schema generation results of the proposed pipeline without CP filtering

Domain Name	Desc Granularity	LLM instance #	Total Combinations	Solved Combinations	Distinct Plan #	Avg. Plan Length	Applied CP Threshold	CP Threshold Value
libraryworld	detailed	7	18144	0	N/A	N/A	False	N/A
libraryworld	detailed	10	86400	4560	7	6.57	False	N/A
libraryworld	ambiguous	7	26460	0	N/A	N/A	False	N/A
libraryworld	ambiguous	10	124416	17976	7	5.14	False	N/A
dungeon	detailed	7	600	600	5	4	False	N/A
dungeon	detailed	10	2800	2099	5	4	False	N/A
dungeon	ambiguous	7	360	180	1	5	False	N/A
dungeon	ambiguous	10	700	140	1	5	False	N/A
minecraft	detailed	7	1323	216	2	3	False	N/A
minecraft	detailed	10	5250	3598	4	3	False	N/A
minecraft	ambiguous	7	7350	2800	5	3	False	N/A
minecraft	ambiguous	10	21600	6000	6	3	False	N/A

Table 8: Detailed action schema generation results of the proposed pipeline WITH CP filtering

Domain Name	Desc Granularity	LLM instance #	Total Combinations	Solved Combinations	Distinct Plan #	Avg. Plan Length	Applied CP Threshold	CP Threshold Value
libraryworld	detailed	7	672	0	N/A	N/A	True	0.398
libraryworld	detailed	10	2160	60	2	5	True	0.398
libraryworld	detailed	15	2340	104	2	5	True	0.398
libraryworld	ambiguous	7	252	0	N/A	N/A	True	0.398
libraryworld	ambiguous	10	3240	560	6	5.3	True	0.398
libraryworld	ambiguous	15	37440	852	10	6.3	True	0.398
dungeon	detailed	7	24	24	3	4	True	0.398
dungeon	detailed	10	48	36	3	4	True	0.398
dungeon	detailed	15	128	96	1	4	True	0.398
dungeon	ambiguous	7	18	9	1	5	True	0.398
dungeon	ambiguous	10	20	4	1	5	True	0.398
dungeon	ambiguous	15	168	108	3	4	True	0.398
minecraft	detailed	7	441	0	N/A	N/A	True	0.398
minecraft	detailed	10	945	630	2	3	True	0.398
minecraft	detailed	15	1287	720	5	3	True	0.398
minecraft	ambiguous	7	2940	1120	6	3	True	0.398
minecraft	ambiguous	10	6720	1920	6	3	True	0.398
minecraft	ambiguous	15	45760	16384	8	2.63	True	0.398

Justification why tested with problems that only need short plans:

Our focus on problems requiring short plans stems from two key reasons:

- **Focus on Schema Set Construction:** The primary challenge in LLM-symbolic planning lies in accurately constructing the action schema set. Once this set is correctly defined, plan generation is handled by the symbolic planner, and the length of the plan becomes no more a crucial factor in evaluating the LLM-symbolic planning pipeline's performance. Our approach excels in generating accurate action schema sets, ensuring reliable plan generation regardless of length.

- **Limitations of Direct LLM Planning Models:** Direct LLM planning models like "Tree of Thoughts" (ToT) suffer from inherent limitations in long-term planning due to their probabilistic nature. Accuracy diminishes exponentially with each step. For example, even with a 99% per-step accuracy, the probability of a correct 100-step plan plummets to 36.6%. This makes direct LLM planners unsuitable for long-term planning. Thus, for fair comparison, we focus on the reasoning capabilities of different approaches under rational plan lengths.

Implication: Our justification highlights a significant advantage of the LLM-symbolic planning pipeline: the quality of plan generation is not affected by the length of the plan, but rather by the quality of the action schema set. This means that our pipeline can be generalized to plan generation of any length. In contrast, direct LLM planning models are fundamentally limited in their ability to guarantee soundness in long-term planning.

Table 9: Detailed Plan for ToT direct planning models in Sussman Anomaly testing case

Model	Plan	Heuristic
ToT GLM	["take-from-table Book2", "place-on-shelf Book2 Book3", "take-from-table Book1", "place-on-shelf Book1 Book2"]	9.0
ToT GLM	["take-from-table Book2", "place-on-shelf Book2 onto Book3", "take-from-table Book1", "place-on-shelf Book1 onto Book2"]	9.0
ToT GLM	["take-from-table Book2", "place-on-shelf Book2 onto Book3", "remove-from-shelf Book1", "place-on-shelf Book1 onto Book2"]	9.0
ToT GPT-3	["take-from-table Book2", "place-on-shelf Book2 Book3", "place-on-table Book1", "place-on-shelf Book1 Book2", "remove-from-shelf Book3", "place-on-shelf Book3 Book2", "place-on-shelf Book1 Book2", "check-out Book1"]	5.11
ToT GPT-3	["take-from-table Book2", "place-on-shelf Book2 Book3", "place-on-table Book1", "place-on-shelf Book1 Book2", "remove-from-shelf Book3", "place-on-shelf Book3 Book2", "remove-from-shelf Book2", "place-on-shelf Book2 Book1"]	5.11
ToT GPT-3	["take-from-table Book2", "place-on-shelf Book2 Book3", "place-on-table Book1", "place-on-shelf Book1 Book2", "place-on-shelf Book1 Book3", "place-on-shelf Book2 Book1", "place-on-shelf Book2 Book3", "remove-from-shelf Book2", "place-on-shelf Book2 Book1"]	4.11
ToT GPT-4	["take-from-table (Book2)", "place-on-shelf (Book2, Book3)", "take-from-table (Book1)", "place-on-shelf (Book1, Book2)"]	8.5
ToT GPT-4	["take-from-table(Book2)", "place-on-shelf(Book2, Book3)", "take-from-table(Book1)", "place-on-shelf(Book1, Book2)"]	8.5
ToT GPT-4	["take-from-table(Book2)", "remove-from-shelf(Book3)", "place-on-table(Book3)", "place-on-table(Book2)", "take-from-table(Book2)", "place-on-table(Book2, Book3)", "take-from-table(Book1)", "place-on-table(Book1, Book2)", "take-from-table(Book1)"]	8.33
ToT GPT-4	["take-from-table(Book2)", "remove-from-shelf(Book3)", "place-on-table(Book3)", "place-on-table(Book2)", "take-from-table(Book2)", "place-on-table(Book1)", "take-from-table(Book3)", "place-on-shelf(Book3, Book1)", "take-from-table(Book1)"]	7.89
ToT GPT-4	["take-from-table(Book2)", "remove-from-shelf(Book3)", "place-on-table(Book3)", "place-on-table(Book2)", "take-from-table(Book3)", "place-on-table(Book2)", "place-on-table(Book3)", "take-from-table(Book2)", "place-on-shelf(Book2, Book3)"]	7.78
ToT GPT-4	["take-from-table(Book2)", "remove-from-shelf(Book3)", "place-on-table(Book3)", "place-on-table(Book2)", "take-from-table(Book3)", "place-on-table(Book2)", "place-on-table(Book3)", "take-from-table(Book2)", "place-on-table(Book3)"]	7.44

Table 10: Detailed Plan for the proposed LLM-symbolic planning pipeline in Sussman Anomaly testing case

Model	Plan	RankScr
Ours GLM	["(remove-from-shelf book3 book1)", "(take-from-table book2)", "(place-on-shelf book2 book3)", "(take-from-table book1)", "(place-on-shelf book1 book2)"]	0.788
Ours GLM	["(take-from-table book2)", "(place-on-shelf book2 book3)", "(remove-from-shelf book3 book1)", "(take-from-table book1)", "(place-on-shelf book1 book2)"]	0.770
Ours GLM	["(remove-from-shelf book3 book1)", "(take-from-table book1)", "(place-on-shelf book1 book2)", "(take-from-table book2)", "(place-on-shelf book2 book3)"]	0.724
Ours GLM	["(remove-from-shelf book3 book1 reference)", "(take-from-table book2)", "(place-on-shelf book2 book3)", "(take-from-table book1)", "(place-on-shelf book1 book2)"]	0.632
Ours GLM	["(remove-from-shelf book3 book1)", "(take-from-table book1)", "(place-on-table book3)", "(check-out book1)", "(take-from-table book2)", "(place-on-table book1)", "(place-on-shelf book2 book3)", "(take-from-table book1)", "(return-book book1)", "(place-on-shelf book1 book2)"]	0.569
Ours GLM	["(take-from-table book2)", "(place-on-shelf book2 book3)", "(remove-from-shelf book3 book1)", "(check-out book1)", "(place-on-shelf book1 book2)"]	0.552
Ours GLM	["(take-from-table book2 reference)", "(place-on-shelf book2 book3)", "(remove-from-shelf book3 book1 reference)", "(take-from-table book1 reference)", "(place-on-shelf book1 book2)"]	0.512

For more detailed generated action schema sets and plans, please go to the following folders in the code repository:

- `data/07_model_output/tree_of_thought_plans`
- `data/07_model_output/llm_to_domain_to_plans`

F.1 Extra Experiments Outside the Main Scope

Evaluating the Quality of Action Schema Generation We evaluate the diversity and solvability of domains, with the quality assessment being conducted solely through plans against the Tree of Thought (ToT) for two primary reasons. Firstly, the absence of schemas for ToT means that the evaluation of the planner can only be carried out at the plan level. Secondly, we do not assume there exists a single ground truth schema set for a given natural language described planning problem. It is because a quality examination using a ground truth (GT) domain contradicts our assumption that a natural language problem description can have multiple interpretations, and that a one-to-one mapping from an ambiguous description to formal formats is inherently flawed, as mentioned on Section 1. To gauge the ambiguity of the description, refer to the `data/01_raw/pddl_domain/*/*/data.py` files, where there is considerable freedom in determining which predicates to include, particularly for layman settings.

Predicate-level evaluation with a GT schema, such as F1, overlooks the overall synergy of individual actions in the modeling process and would be biased towards a single perspective, as discussed on the limitation of the current single expert-in-loop approach. Our objective is to make planning systems more accessible to a wider range of users, which renders predicate-level quality evaluation an indirect measure of success. For the purpose of completeness, we provide the evaluation results against the GT domain in Table 11.

Table 11: Evaluating the accuracy of generated action schemas against the ground truth domain data involves a predicate-level analysis, which yields metrics such as precision, recall, and the F1 score. Due to the inherent ambiguity of natural language descriptions, the generated schema set will not perfectly align with the ground truth schema set. Furthermore, the evaluation results fail to reflect the final quality of the generated plans within our hybrid LLM planning pipeline.

Info Granularity	Eval w.r.t. GT domain		
	Precis.	Recall	F1
Layman	0.696	0.617	0.654
Detailed	0.679	0.636	0.657

It is evident in Table 11 that the schema set generated from natural language descriptions will not perfectly align with the ground truth schema set. The primary reason for this discrepancy lies in the inherent ambiguity of natural language, which cannot precisely capture the exact intentions of the original schema author. Consequently, metrics such as precision, recall, and F1 score are inadequate to assess the quality of the generated plans within our hybrid Large Language Model (LLM) planning pipeline. Therefore, the evaluation of the quality of the generated action schema in relation to the ground truth model is considered a secondary measure of assessment.

To see how ambiguous the natural language descriptions are, we are posting the following examples of the natural language descriptions of the actions for the barman domain:

```
ACTION_DESC_DICT = {
    'clean-shaker' : {
        "layman": "The robot cleans an empty shaker.",
        "detailed": "Clean the empty shaker. The action is related to whether the shaker is
    empty and whether barman holds the shaker.",
    },
    'clean-shot' : {
        "layman": "The robot cleans a used shot glass.",
        "detailed": "clean the shot, it is depending on the conditions that whether the shot
    is empty and the precondition also need to know about whether barman holds the container.
    ",
    },
    'empty-shaker' : {
        "layman": "The robot empties a shaker that has been shaken, changing its level.",
        "detailed": "Pour the contents out of the shaker. Things that are related to this
    action are whether the shaker is shacked or not, whether the shaker contains the cocktail
     or not and whether the shaker level goes back to empty level.",
    },
    'empty-shot' : {
        "layman": "The robot empties a shot glass it's holding.",
        "detailed": "empty the shot. It depends on whether the container has beverage and
    whether barman holds the container.",
    },
    'fill-shot' : {
```

```
        "layman": "The robot fills an empty, clean shot glass with an ingredient from a
    dispenser.",
        "detailed": "use a hand to hold a clean shot and fill ingredient that comes from the
    dispenser.",
    },
    'grasp' : {
        "layman": "The robot uses a hand to pick up a container from the table.",
        "detailed": "The action that barman is grasping is depending on the precondition that
     whether the container is in barman's hands or on the table.",
    },
    'leave' : {
        "layman": "The robot places a container it's holding back onto the table.",
        "detailed": "The action that barman is going to leave their container from his hand
    on to table.",
    },
    'pour-shaker-to-shot' : {
        "layman": "The robot pours a beverage from a shaker into an empty, clean shot glass."
    ,
        "detailed": "Pour the shaken alcohol into a clean empty shot glass. The action is
    related to whether the shot is empty and clean, and whether the barman holds the shot and
     also it will affects the shaker level. ",
    },
    'pour-shot-to-clean-shaker' : {
        "layman": "The robot pours an ingredient from a shot glass into a clean shaker,
    changing its level.",
        "detailed": "Pour the hard liquor and other ingredients into a clean empty shaker and
     shake. The action is depending on the conditions that whether the shaker is empty and
    clean and also whether barman holds and shakes the shaker.",
    },
    'pour-shot-to-used-shaker' : {
        "layman": "Similar to the previous action, but the shaker already contains
    ingredients and isn't clean.",
        "detailed": "Pour the hard liquor and other ingredients into the used shaker. It
    depends on whether the barman start shaking or not. Also you need to hold the shot.",
    },
    'refill-shot' : {
        "layman": "The robot refills a shot glass with the same ingredient it previously
    contained.",
        "detailed": "refill the shot with the same ingredient. The action is depending on the
     conditions that whether the shot is empty and used and whether barman holds the shot.",
    },
    'shake' : {
        "layman": "The robot shakes a shaker containing two ingredients to make a cocktail.",
        "detailed": "make cocktail with two ingredients. The action that whether barman
    shakes the shaker is depending on the preconditions that whether the shaker has cocktail
    or ingredient and whether barman holds and shakes the shaker.",
    },
}
```

The "barman" PDDL domain snippets of the PDDL expert are as follows:

```
(define (domain barman)
(:requirements :typing :strips)
(:types
    hand level beverage dispenser container - object
    ingredient cocktail - beverage
    shot shaker - container
)
(:predicates
    (ontable ?c - container)
    (holding ?h - hand ?c - container)
    (handempty ?h - hand)
    (empty ?c - container)
    (contains ?c - container ?b - beverage)
    (clean ?c - container)
    (used ?c - container ?b - beverage)
```

```
        (dispenses ?d - dispenser ?i - ingredient)
        (shaker-empty-level ?s - shaker ?l - level)
        (shaker-level ?s - shaker ?l - level)
        (next ?l1 ?l2 - level)
        (unshaked ?s - shaker)
        (shaked ?s - shaker)
        (cocktail-part1 ?c - cocktail ?i - ingredient)
        (cocktail-part2 ?c - cocktail ?i - ingredient)
    )

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

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

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

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

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

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

```
        (clean ?s))
)

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

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

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

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

(:action shake
    :parameters (?b - cocktail ?d1 ?d2 - ingredient ?s - shaker ?h1 ?h2 - hand)
    :precondition (and (holding ?h1 ?s)
        (handempty ?h2)
        (contains ?s ?d1)
        (contains ?s ?d2)
        (cocktail-part1 ?b ?d1)
        (cocktail-part2 ?b ?d2)
```

```
            (unshaked ?s))
        :effect (and (not (unshaked ?s))
            (not (contains ?s ?d1))
            (not (contains ?s ?d2))
            (shaked ?s)
            (contains ?s ?b))
    )

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

It is evident that natural language descriptions possess inherent ambiguity, especially the layman setting. A direct one-to-one correspondence with the ground truth schema might fail to account for this ambiguity and the various valid interpretations that can arise from such descriptions. This recognition is the cornerstone of our research, as detailed in Section 1. Consequently, we advise against relying on the ground truth schema for evaluating the quality of the generated action schema sets.

In the context of the "barman" example, we found that layman texts frequently omit predicates related to hand movements while still preserving the essential semantics of the action schema. As a result, LLMs are also likely to produce action schemas that omit the details about the hand interaction, thereby not strictly matching the ground truth schema. Importantly, even without the inclusion of hand predicates, these generated schemas can still remain solvable and capable of generating plans that align with user preferences. Therefore, relying solely on the ground truth schema for evaluating the quality of the generated action schemas may not be an effective measure, as it does not fully consider the flexibility inherent in natural language descriptions.

Cost Analysis Conducting a comprehensive cost analysis for the proposed pipeline presents a significant challenge. When analyzing the costs associated with our pipeline against those of an expert-in-the-loop approach, several critical factors must be taken into account: (1) the consultant fees for expert involvement, and (2) the time spent for experts to identify and interactively correct errors. Quantifying the cost of our pipeline is further complicated by its dependence on the number of action schema combinations, the complexity of the planning tasks, and the specifications of the CPU used for the symbolic planner.

Moreover, we need to understand that the Tree-of-Thoughts (ToT) approach, which relies solely on LLM text generation for planning, **also** becomes more computationally expensive as the tree's depth and breadth expand. This complexity leads to the fact that previous studies on hybrid planning pipelines, such as (Guan et al. 2023; Liu et al. 2023), have not provided a comprehensive cost analysis either.

Given these considerations, we recognize the importance of conducting a detailed cost analysis in future work to provide a more comprehensive cost comparison between our pipeline and other competing approaches.