Generating Domain Models for Automated Planning from Natural Language Descriptions

Giacomo Acitelli[1,2], Andrea Marrella[1], Jacopo Rossi[1], Giada Troilo[1] and Han van der Aa[2]

***Abstract*— The accuracy of automated planning tasks depends on the quality of the PDDL domains models used to represent them; however, manually developing these models is complex and prone to errors. While Large Language Models (LLMs) can process natural language (NL) descriptions of planning tasks, generating accurate and complete PDDL domains remains a significant challenge. In this paper, we propose a modular approach for constructing PDDL domains from NL descriptions through a series of structured extraction and refinement steps involving LLMs. Experiments with expert human evaluators and an LLM-as-a-judge on 30 benchmark planning tasks from the TEXT2WORLD dataset show that our approach generates syntactically correct and semantically coherent domains without the need for structured NL inputs, demonstrating its robustness and generality.**

I. INTRODUCTION

Automated Planning (AP) is a branch of Artificial Intelligence (AI) that addresses the problem of generating a course of actions (i.e., a *plan*) to achieve a goal, given a description of the domain of interest and an initial state. *Planning Domain Definition Language* (PDDL) [1] is the defacto standard language for representing AP models, which separates the general rules of the environment (i.e., the *domain*) from specific instances (i.e., the *problem*). In recent years, the planning community has developed numerous *planners* with effective search heuristics, capable of scaling to large problems and solving challenging issues across various Computer Science domains [2]. Nonetheless, the manual creation of PDDL domains is a labor-intensive and error-prone knowledge engineering challenge, which significantly limits the adoption of planning in real-world applications [3].

In this context, Large Language Models (LLMs) have recently been explored as tools for planning formalization, aiming to bridge the gap between *unstructured* natural language (NL) descriptions of planning tasks and *structured* PDDL representations [4]. However, LLM-generated PDDL domains often contain syntactic errors, inconsistent type assignments, or incomplete predicate and action specifications [5]. These shortcomings arise from the inherent ambiguity of NL, the rigid syntactic requirements of PDDL, and the lack of intermediate reasoning steps to help LLMs capture the implicit logical dependencies in planning tasks [6], [7].

In this paper, we tackle this challenge by proposing a *multi-step approach* that extracts the key components of a planning task (e.g., object types, predicates, actions, etc.) from a NL description through sequential interactions with an LLM. After each interaction, our approach generates an intermediate PDDL representation that incorporates the requested components, progressively refining it until a complete, syntactically correct and logically consistent PDDL domain is produced. These properties are ensured by structuring the intermediate LLM outputs with BAML[1], a prompting framework that defines explicit schemas for input-output mappings. BAML enables static validation at each step with a PDDL syntax validator, ensuring control, consistency and reproducibility throughout the generation process [8].

To demonstrate the ability to generate valid PDDL domains, we evaluate our approach using 30 planning task descriptions from the well-known TEXT2WORLD benchmark [9], which covers a wide range of planning complexities and PDDL requirements. Our evaluation involves both *expert humans* and an *LLM-as-a-judge* approach [10]. The results show strong agreement between the two, confirming that the generated PDDL domains are correct, expressive, and actionable. We note that, unlike previous NL-to-PDDL methods using LLMs [4], our approach does not depend on structured inputs that predefine the generation of the final PDDL domain. Conversely, by relying solely on the NL description of the planning task and structuring each intermediate LLM's output with BAML, we ensure reliable results across diverse planning tasks.

The rest of the paper is organized as follows. Section II covers background from AP and LLMs, and discusses related work. Section III details our approach, evaluated in Section IV. Finally, Section V concludes the paper.

II. BACKGROUND AND RELATED WORK

A. Automated Planning

Automated Planning (AP) is a research field focused on developing model-based autonomous solutions in AI [11]. In this respect, planning systems (i.e., *planners*) are problem-solving algorithms that operate on explicit representations of states and actions. PDDL [1] is the standard Planning Domain Definition Language; it allows us to formulate a *planning model* with the description of the initial state of the world, the desired goal condition, and the planning domain.

A *planning domain* is built from a set of *propositions* describing the state of the world (i.e., the set of propositions that are true) and a set of *actions* Ω that can be executed. An *action schema* $a \in \Omega$ defines the list of *input parameters* for a, the *preconditions* under which a can be executed, and the *effects* of a on the state of the world. Both preconditions and effects are stated in terms of propositions in the planning

[1]Sapienza University of Rome, Rome, Italy
[2]Faculty of Computer Science, University of Vienna, Vienna, Austria

[1]https://github.com/boundaryml/baml

domain, represented as *boolean predicates* and *numeric fluents*. While predicates represent facts about *domain objects* (i.e., things in the world that interest us) that can be true or false, fluents are number-valued functions that can be used to specify how the action increases the total cost of the plan being computed. PDDL allows for advanced propositional action declarations, including *negated preconditions* and *universal/existential quantification* of objects. The effects of actions can also be *conditional* (when-effects). In addition, PDDL includes the ability to *type* the parameters that appear in actions, predicates, and numeric fluents. In a state of the world, only actions whose preconditions are fulfilled can be executed. The values of propositions in the planning domain can change as a result of the execution of actions, which, in turn, lead the planning domain to a new state.

There exist several forms of planning in the AI literature (e.g., to capture durative and nondeterministic actions, partial observability, etc., cf. [12]). In this paper, we focus on *classical planning* techniques characterized by *fully observable* and *deterministic* domains. To encode a classical planning problem, we use PDDL 2.1 [1], which is considered as the most significant evolution over the original version of the language (called STRIPS) and is nowadays supported by the majority of state-of-the-art domain-independent planners.[2] A solution to a planning problem is a sequence of operators—a *plan*—whose execution transforms the initial state into a state satisfying the goal by optimizing a pre-specified metric, generally related to the minimization of the plan's total cost.

B. LLMs and Prompt Engineering

Large Language Models (LLMs) like GPT, Gemini, and LLaMA have become essential tools in modern computational workflows, supporting reasoning, programming, and decision-making across various domains. These models are widely applied in areas such as code generation [13], information extraction [14], and the creation of structured knowledge representations [15]. Their growing adoption has transformed how work is carried out by positioning LLMs as versatile interfaces between NL and formal systems.

In this context, it is important to emphasize the role of *prompt engineering*, which involves designing and refining textual instructions to improve the accuracy of an LLM's outputs. A prompt serves as task-specific guidance, helping the LLM understand intent, reduce ambiguity, and meet explicit constraints. The wording of a prompt significantly influences LLM behavior, and underspecified prompts often fail in knowledge-intensive tasks [16].

A recent study [17] identified 39 prompting methods, highlighting the increasing importance of structured prompting strategies such as Chain-of-Thought, ReAct, and Tree-of-Thoughts. Despite this progress, prompting remains largely procedural (with the LLM being instructed *how* to act), relying on linguistic phrasing, implicit heuristics, and user-driven trial-and-error, rather than formal behavioral specification [18]. As a consequence, small lexical variations can lead to relevant divergences in LLM reasoning, resulting in unpredictable outcomes across LLM sessions [19].

To address this gap, recent works have introduced many frameworks [19] that formalize LLM interactions as *programmable* and *declarative* processes (specifying *what* the LLM should achieve), preserving the semantic integrity of the prompts. Among these frameworks, BAML (Boundary AI Markup Language) treats prompts as strongly-typed functions with defined inputs and an expected output schema (e.g., typed JSON). BAML enables the definition of behavioral requirements, such as tone regulation and format consistency, without dictating implementation details. BAML incorporates a robust parsing and error-correction layer that converts the LLM's raw, approximate output into a precise structured data model. If the output deviates from the expected schema, BAML raises actionable errors, simplifying debugging and allowing developers to apply software engineering practices like type checking and static analysis [20].

In this paper, building on these considerations, we present a hybrid approach to prompt engineering that combines *chain-of-thought* prompting with BAML to generate accurate PDDL planning domains from NL descriptions. Our approach eliminates the need for structured inputs or human intervention, addressing one of the key limitations found in previous literature works.

C. Related Work

Recent work has explored using LLMs to derive planning domains from NL. Approaches such as NLtoPDDL [21], PDDLEGO [22] and PROC2PDDL [23] show that LLMs can extract action parameters, preconditions, and effects from planning task descriptions. The TEXT2WORLD benchmark [9] introduces a large set of PDDL domains paired with structured NL descriptions and evaluates generated domains using multi-criteria metrics including executability and structural similarity. However, results show that PDDL domain generation remains challenging, as generated domains often contain incomplete or partially specified actions [23], [9].

Another line of work focuses on LLM-assisted domain-modeling pipelines. Guan et al. [24] incrementally constructs PDDL domains from NL action descriptions through a generate–test–critique approach, combining LLM outputs with feedback from human experts. Evaluations report that inaccuracies in generated models typically trigger several refinement cycles and require significant expert supervision [5], [25]. Huang et al. [25] point out that expert-in-the-loop refinement constitutes a scalability bottleneck and propose a library of alternative models combined with an automatic semantic filtering and ranking step to approximate expert feedback. Oswald et al. [26] evaluate LLM-generated domains using plan-equivalence metrics to compare reconstructed models with ground-truth specifications, showing that even strong models achieve only moderate reconstruction quality.

Another research direction explores prompting strategies for extracting structured information from NL using LLMs. Neuberger et al. [8] propose a general strategy based on explicit task decomposition, a meta-language defining the target

[2] http://icaps-conference.org/index.php/main/competitions

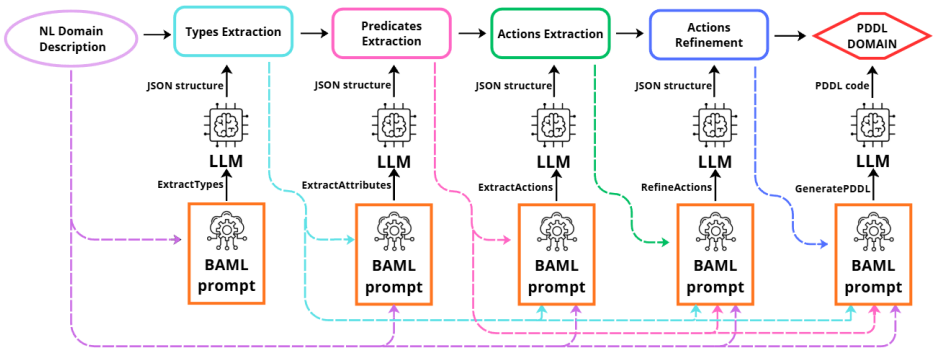

Fig. 1. Modular Architecture for PDDL Generation (NL → PDDL) using BAML and LLM.

constructs, and strict output-format constraints. Their results show that stepwise extraction with schema-based control improves consistency and reduces structural errors. While developed for process-model extraction, these principles are broadly applicable to NL-to-model pipelines and align with the design of our approach.

The novelty of our approach compared to previous work is twofold. First, unlike PROC2PDDL and TEXT2WORLD [23], [9], we require only NL task descriptions as input, without assuming pre-defined predicate lists, domain headers, or action names (assumptions that are often unrealistic). Second, rather than relying on intermediate human intervention, we employ a strictly modular pipeline with schema validation, extracting types, predicates, and actions in separate stages under BAML-enforced structures, which mitigates common issues such as predicate mismatches and incomplete or inconsistent action schemas.

III. APPROACH

Our approach defines a structured pipeline for generating PDDL domain models from NL descriptions. As shown in Fig. 1, it consists of six sequential stages: (*i*) *Input Description*; (*ii*) *Types Extraction*; (*iii*) *Predicates Extraction*; (*iv*) *Actions Extraction*; (*v*) *Actions Refinement* and (*vi*) *PDDL Domain Generation*. Each stage leverages chain-of-thought prompting combined with BAML schema constraints to maintain well-typed and consistent intermediate representations [8]. We next describe each stage using a running example based on the classical Blocks World domain.

Input Description. The input is a free-form NL description of the domain to be modeled. For example, the following is a potential NL description of the Blocks World domain: "*The Blocks World domain involves manipulating a set of blocks identified by different labels. Blocks can be placed either on the table or on top of other blocks. Only one block at a time can be stacked on another block, and a block can be moved only if it is free, i.e., if no other block is placed on top of it. A robotic arm is the only agent capable of performing movements, by picking up blocks and placing them on the table or onto other blocks.*"

Types Extraction. The first stage extracts the object types of the planning domain from the NL description. Establishing a consistent type hierarchy is essential for PDDL modeling, as it reduces ambiguity in the subsequent stages and enables typed representations of predicates and actions. This step uses a structured prompt (Prompt 1) to guide the LLM toward PDDL-compliant type definitions while allowing the inference of implicit hierarchies.

Prompt 1: Types Extraction

Task: Transform a NL scenario into a structured JSON that includes admissible object types.

Rules: Identify the distinct domain objects, enforce PDDL naming conventions (lowercase, hyphenated, etc.), infer implicit hierarchies (e.g., 'agent'), and ensure that all parent types are defined.

Output: JSON object containing the extracted information.

Applied to the Blocks World description, this stage produces a compact set of domain types, including *block* (stackable objects), *table* (support surface), and *robotic-arm* (the manipulating agent). These types establish the foundation for subsequent predicate and action extraction.

Predicates Extraction. After identifying object types, the next stage extracts boolean predicates (and numeric fluents when needed) that capture object properties and relations. In PDDL, predicates define the admissible state space and serve as the interface through which actions affect the world. This step is guided by (Prompt 2), which takes the NL description and the typed objects from the previous stage as input:

Prompt 2: Predicates Extraction

Task: Extract boolean predicates and numeric fluents for each type.

Rules: Enforce single typing for each parameter. Create separate predicates and fluents for each distinct type combination, using unique and descriptive names.

Output: JSON object containing the extracted predicates and fluents.

The output links each object type to the predicates and fluents that are supported by the description. This stage is prone to two common issues: generating PDDL-invalid constructs and collapsing distinct relations into overly general predicates. To mitigate these errors, the prompt enforces strict typing (one type per parameter) and mandates separate

predicates/fluents for different type combinations.

In the Blocks World example, the resulting output includes *on-table*, *on-block*, *clear*, and *held-by-arm* for a *block* object, while *robotic-arm* is characterized by *arm-empty*. No predicates are associated with *table*, reflecting its passive role.

Actions Extraction. This stage captures the domain dynamics by extracting state transitions and representing them as PDDL actions. For each action, the LLM determines a name, typed parameters, preconditions, and effects. The step takes as input the NL description along with the types and predicates/fluents from previous stages:

Prompt 3: Actions Extraction

Task: Extract actions with parameters, preconditions, and effects.

Rules: Include needed parameters and infer inverse actions when implied by the NL description. Effects must be explicitly specified using add/delete for predicates and increase/decrease for fluents, with consistent use of predicate and fluent names.

Output: JSON object containing the extracted action schemas.

This prompt guides the definition of actions using the previously extracted predicates and fluents. Effects are specified to indicate which predicates an action makes true (*add*) or false (*delete*) and how numeric fluents are incremented (*increase*) or decremented (*decrease*). It also allows defining inverse actions when required by the NL description.

In the Blocks World example, the generated action set consists of four well-typed actions performed by the *robotic-arm*: *pick-up-from-table* and *put-down-on-table* handle transfers between the table and the arm, while *stack* and *unstack* manage stacking and unstacking of blocks.

Actions Refinement. While Action Extraction produces explicit PDDL action schemas, the resulting actions can still be (i) *incomplete*, missing implicit preconditions or effects needed for planning, or (ii) *overly specific*, generating duplicate actions instead of a compact, parameterized set. To address this, we introduce an Action Refinement stage that reviews the extracted actions to improve semantic completeness while maintaining PDDL compliance. This refinement is guided by a prompt that considers the full domain context, i.e., the NL description, the extracted types, predicates and fluents, and the preliminary action schemas:

Prompt 4: Actions Refinement

Task: Refine the extracted actions by completing implicit preconditions and effects required for problem solvability.

Rules: Preserve all actions. Infer constraints based on domain semantics (e.g., adding clearance checks). Detect necessity for universal or existential quantifiers, and conditional effects.

Output: JSON object containing the refined actions.

This step ensures consistency across domain components, adds missing semantic constraints, and flags constructs beyond STRIPS (e.g., quantifiers). Actions are preserved and merged only if semantically equivalent. It is most useful for generic or underspecified domains, and (typically) does not alter well-structured domains like Blocks World.

PDDL Domain Generation. This stage combines the intermediate outputs from the previous stages into a planner-ready PDDL domain file consistent with the NL description:

Prompt 5: PDDL Domain Generation

Task: Generate a complete, valid, and executable PDDL domain file.

Rules: Ensure syntactic correctness and generate a planner-ready PDDL domain file, declaring only the requirements supported by the extracted model.

Output: PDDL domain file with all extracted components.

This step ensures planner compatibility by declaring only the PDDL requirements needed by the extracted model: *:strips* and *:typing* are always included, while others (e.g., negative preconditions or numeric fluents) are added only if required. A syntax check is performed using VAL[3].

The complete prompts and outputs summarized in this section are available for investigation in the appendix, available at `https://github.com/giacomo1096/NL2PDDL`.

IV. Evaluation

We evaluated our approach using both quantitative and qualitative analyses. Quantitatively, generated PDDL domains were compared to 30 Text2World gold models [9] using structural metrics (e.g., numbers of types, predicates/fluents, actions, etc.). Qualitatively, modeling adequacy was assessed against the NL descriptions using an LLM-as-a-judge alongside independent human expert reviews. The 30 gold models were randomly selected from those providing clear and understandable NL descriptions suitable for interpretation by human experts.

Experimental Setup. We followed an evaluation protocol consistent with the Text2World benchmark but under a more constrained setting: our approach uses only NL descriptions as input, without any structured hints (e.g., predicate lists or action names). We tested it on 30 gold-standard planning tasks using the original NL descriptions in a fully automatic, end-to-end setup with no human intervention. All experiments were conducted using *Gemini 2.5 Pro*. Because our input setting is more challenging than standard Text2World configurations, we use the benchmark primarily for structural and semantic analysis rather than for direct score comparisons. The code to reproduce the experiments and the obtained results are available at: `https://github.com/giacomo1096/NL2PDDL`.

Quantitative Evalution. We quantitatively evaluated the formal validity and structural properties of the PDDL artifacts generated by our pipeline, comparing them to the corresponding Text2World gold models. First, we checked syntactic correctness: all generated domain files pass VAL and ENHSP parsing and syntax checks [27], showing that our

[3]`https://github.com/KCL-Planning/VAL`

Domain	Types		Pred./Fluents		Actions		Avg #Param.	
	T2W	Ours	T2W	Ours	T2W	Ours	T2W	Ours
Books	2	2	7	6	1	5	2	2.7
Cleaning	2	2	7	3	6	2	3	2.5
Dock Worker	3	4	5	6	3	3	3	3.66
Elevator	10	9	6	23	3	15	2.66	2.4
Grid	0	6	12	9	5	4	2.6	4.25
Gripper	0	5	7	5	2	3	2.5	3
Hanoi	0	2	3	7	1	4	3	2
Hiking	1	4	6	7	2	4	2	2
Ice Cream	0	2	13	4	7	4	0.71	1.25
Maintenance	3	4	4	3	1	3	2	3.33
MineWorld	6	9	7	8	6	5	3.33	3.2
Mining	1	10	11	12	7	9	1.42	3.44
MPrime	6	5	8	20	4	7	5.25	3.55
Network	4	7	13	16	5	7	6	3.71
Painting	3	4	11	3	7	3	3.28	3.67
Parking	2	5	6	13	4	8	3	5.5
Peg Solitaire	1	4	6	6	3	3	2.33	4.33
Puzzle	0	4	8	5	4	4	3	5
Rovers	7	10	25	16	9	8	4	3.25
Sandwich	7	6	13	8	6	7	2.66	3
Sat Solvers	0	2	4	6	4	2	2.75	1
Satellite	4	6	8	11	5	8	2.8	2.62
Scanalizer3D	2	2	7	3	4	6	6	3.83
Snake	1	6	8	13	3	6	3.33	3.66
Sokoban	3	4	4	5	2	2	4	5
Spanner	5	5	6	7	3	4	3.33	3.22
Taxi	3	3	5	4	3	3	3	3
Videogame	11	14	9	22	8	15	4.5	2.8
Visit	1	3	3	3	1	2	2	2.5
Waiter	6	6	6	11	4	5	3.5	3.8

TABLE I

DOMAIN COMPLEXITY METRICS COMPARISON

approach reliably produces planner-parseable PDDL domains without manual intervention.

We then evaluated domain complexity on the 30 gold models using standard metrics: number of types, predicates/numeric fluents, actions, and average parameters per action, capturing differences in model size and abstraction. Since our approach generates PDDL domains directly from NL without predefined predicate or action inventories, deviations from the gold models often reflect valid alternative modeling choices, such as explicitly representing textual details. Overall, the extracted type and predicate sets largely match the gold domains. Additionally, in over half of the domains, at least one gold action schema is more abstract or underspecified than the corresponding action produced by our pipeline, reflecting the more structured nature of our approach. Table I reports the detailed results.

Qualitative Evaluation. For the qualitative evaluation, we employed a dual strategy combining an automatic LLM-as-a-judge approach with expert assessments by planning specialists. These perspectives are complementary: the LLM enables scalable and reproducible semantic evaluation, while human experts provide deeper, domain-aware judgments.

LLM-as-a-judge: The LLM-as-a-judge approach uses LLMs to perform structured semantic evaluation of the generated PDDL domains, going beyond syntactic validity to assess consistency with the original NL description, following established best practices [10]. For each task, an external LLM is given the NL description, the generated PDDL domain, and a strict JSON-based scoring schema covering three dimensions: predicate accuracy (30 points), action logic and common-sense compliance (35 points), and generality and sufficiency for solvability (35 points). Alongside an overall score from 0 to 100, the judge must provide a detailed rationale for any deductions, explicitly citing issues in predicates, preconditions, or effects. This structured prompting approach (detailed in Prompt 6) ensures the evaluation is both measurable and automatable. In our experiments, we used GPT-4.1 and Gemini 2.5 Pro as judges and report results averaged across the two models.

Prompt 6: LLM-as-a-judge

Role: Expert in PDDL and NL understanding, acting as an evaluator.

Task: Provide a quantitative assessment (0-100 points) of how well the generated PDDL domain captures the generalized rules, entities, and actions from the provided NL description.

Rules: Assess Predicate Accuracy ($30 pts$), Action Logic and Common Sense Compliance ($35 pts$), and Generality/Sufficiency ($35 pts$) for solvability. Must justify all deductions and reference specific issues in the PDDL.

Output: JSON object containing the *Domain_Modeling_Score* ($0-100$) and a detailed *Rationale*.

Expert Evaluation: In parallel, we conducted a qualitative evaluation with human experts in automated planning. Experts assigned scores from 0 to 100 using the same three dimensions as the LLM-as-a-judge for comparability. First, they evaluated the predicate set (30 points) to verify that predicates accurately capture key properties and relations. Next, they assessed the action schemas (35 points), checking preconditions and effects for internal consistency. Finally, they judged generality and sufficiency (35 points), evaluating whether the domain supports all behaviors needed to solve the NL-described tasks. Each domain was independently evaluated by five experts, and final scores were averaged across their assessments.

Discussion: Table II reports a comparison between the scores assigned by the LLM-as-a-judge and those provided by human experts. Overall, the two evaluations show strong alignment, indicating that the automatic assessment reliably captures key aspects of semantic correctness and modeling quality. Minor discrepancies typically arise in domains where multiple valid abstraction choices exist, reflecting different modeling preferences. Higher scores are consistently observed for well-structured and less ambiguous domains such as Cleaning, Grid, Puzzle, Satellite, and Sokoban, where both LLMs and experts converge on near-perfect evaluations. Conversely, lower scores appear in domains such as Books, Elevator, MPrime, and Videogame, which involve richer interaction semantics or implicit abstractions that admit multiple valid modeling choices. Overall, these results show that our approach reliably produces semantically coherent, usable, and text-faithful PDDL domains, and that structured extraction with schema validation effectively reduces common semantic errors in NL-to-PDDL generation.

V. CONCLUSION AND FUTURE WORKS

In this paper, we presented a schema-guided approach for generating PDDL domain models directly from unstructured NL descriptions. By decomposing the NL task

Domain	LLM-as-a-Judge	Expert Judges
Books	63.5	69
Cleaning	98.5	98
Dock Worker	93.5	95
Elevator	51.5	60
Grid	97.5	98
Gripper	90	95
Hanoi	90	98
Hiking	96	97
Ice Cream	67.5	67
Maintenance	96	96
MineWorld	76	78
Mining	96	98
MPrime	58.5	60
Network	76	80
Painting	90	90
Parking	76.5	78
Peg Solitaire	87.5	87
Puzzle	98.5	97
Rovers	96.5	97
Sandwich	86	87
Sat Solvers	82	80
Satellite	98.5	98
Scanalizer3D	83.5	80
Snake	71	75
Sokoban	96	97
Spanner	97.5	98
Taxi	96	97
Videogame	63.5	65
Visit	97.5	100
Waiter	87.5	87

TABLE II

COMPARISON BETWEEN LLM AND EXPERTS' SCORES

description and enforcing explicit schemas at each stage, our method produces syntactically correct and semantically coherent planning domains without relying on predefined predicates, actions, or domain-specific hints. Evaluation on the Text2World benchmark demonstrates that this stepwise approach can reconstruct domain models with both structural complexity and semantic accuracy.

The proposed approach has some limitations. Underspecified or ambiguous NL descriptions can lower domain quality, and the current implementation is restricted to classical PDDL 2.1 with typing and numeric fluents, lacking support for temporal or durative constructs, which limits its applicability to non-temporal domains. In this direction, future work will focus on extending the scope of the approach by adding support for temporal and durative fragments of PDDL 2.1, enabling the modeling of time-dependent domains. Additionally, incorporating self-correction and refinement mechanisms for underspecified inputs could enhance robustness and reduce ambiguity in challenging scenarios beyond the one tested in the experiments.

In conclusion, our approach suggests that structured, schema-enforced extraction provides a principled foundation for NL-to-PDDL domain modeling, bridging the gap between informal NL descriptions and formally grounded PDDL representations, and offering a promising solution for advancing knowledge engineering practices in AP.

Acknowledgments. This work has been supported by the Sapienza project FOND-AIBPM, the MUR project PE0000013-FAIR, and by Thales Alenia Space and Regione Lazio, through the fellowships 35757-22066DP000000041-A0627S0031 *Advanced Software Based on Cloud Computing and Machine Learning for Space Systems*.

REFERENCES

[1] M. Fox and D. Long, "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains," *JAIR*, vol. 20, 2003.
[2] A. Marrella, "Automated Planning for Business Process Management," *J. Data Semant.*, vol. 8, no. 2, 2019.
[3] T. L. McCluskey *et al.*, "Engineering knowledge for automated planning: Towards a notion of quality," in *K-CAP'17*, pp. 14:1–14:8, 2017.
[4] M. Tantakoun *et al.*, "LLMs as Planning Formalizers: A Survey for Leveraging Large Language Models to Construct Automated Planning Models," in *ACL 2025*, pp. 25167–25188, 2025.
[5] M. Tantakoun *et al.*, "LLMs as planning modelers: A survey for leveraging large language models to construct automated planning models," in *AAAI 2025 Workshop LM4Plan*, 2025.
[6] S. Kambhampati *et al.*, "LLMs can't plan, but can help planning in LLM-modulo frameworks," *arXiv:2402.01817*, 2024.
[7] K. Valmeekam *et al.*, "Planning in strawberry fields: Evaluating and improving the planning and scheduling capabilities of LRM O1," *arXiv:2410.02162*, 2024.
[8] J. Neuberger *et al.*, "A universal prompting strategy for extracting process model information from natural language text using large language models," in *ER'24*, pp. 38–55, Springer, 2024.
[9] M. Hu *et al.*, "Text2world: Benchmarking large language models for symbolic world model generation," *arXiv:2502.13092*, 2025.
[10] H. Li *et al.*, "LLMs-as-judges: A comprehensive survey on llm-based evaluation methods," *arXiv:2412.05579*, 2024.
[11] H. Geffner and B. Bonet, "A Concise Introduction to Models and Methods for Automated Planning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 8, no. 1, 2013.
[12] P. Haslum *et al.*, *An Introduction to the Planning Domain Definition Language*. Springer, 2019.
[13] A. Fan *et al.*, "Large language models for software engineering: Survey and open problems," in *ICSE-FoSE'23'*, pp. 31–53, IEEE, 2023.
[14] F. Bianchini *et al.*, "Automating industrial quality control: A multimodal llm and rag framework for anomaly detection," in *AIAI'25*, pp. 253–266, Springer, 2025.
[15] F. Bianchini *et al.*, "Enhancing complex linguistic tasks resolution through fine-tuning LLMs, RAG and knowledge graphs (short paper)," in *CAiSE*, pp. 147–155, Springer, 2024.
[16] S. Atreja *et al.*, "What's in a prompt?: A large-scale experiment to assess the impact of prompt design on the compliance and accuracy of llm-generated text annotations," in *AAAI Conf. on Web and Social Media*, vol. 19, pp. 122–145, 2025.
[17] S. Vatsal and H. Dubey, "A survey of prompt engineering methods in large language models for different NLP tasks," *arXiv:2407.12994*, 2024.
[18] M. Calamo and J. Rossi, "Programming large language models," in *Engineering Information Systems with Large Language Models*, pp. 111–137, Springer, 2025.
[19] M. K. Heris, "Prompt Decorators: A Declarative and Composable Syntax for Reasoning, Formatting, and Control in LLMs," *arXiv:2510.19850*, 2025.
[20] M. X. Liu *et al.*, ""we need structured output": Towards user-centered constraints on large language model output," in *CHI*, pp. 1–9, 2024.
[21] S. Miglani and N. Yorke-Smith, "Nltopddl: one-shot learning of pddl models from natural language process manuals," in *KEPS'20*, 2020.
[22] L. Zhang *et al.*, "Pddlego: Iterative planning in textual environments," *arXiv:2405.19793*, 2024.
[23] T. Zhang *et al.*, "Proc2pddl: Open-domain planning representations from texts," *arXiv:2403.00092*, 2024.
[24] L. Guan *et al.*, "Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning," in *NeurIPS'23*, 2023.
[25] S. Huang *et al.*, "Planning in the dark: Llm-symbolic planning pipeline without experts," in *AAAI*, vol. 39, pp. 26542–26550, 2025.
[26] J. Oswald *et al.*, "Large language models as planning domain generators," in *ICAPS'24*, vol. 34, pp. 423–431, 2024.
[27] E. Scala, "ENHSP: Expressive Numeric Heuristic Search Planner." `https://sites.google.com/view/enhsp/`, 2023.