Assessing LLM Performance in Structured Planning Translation

Akash Sivakumar
1229716620
MS CS
asivak21@asu.edu

Aditi Ganapathi
1229975723
MS CS
aganap12@asu.edu

1 Introduction

There are several areas where Large Language Models (LLMs) remain unreliable even with their tremendous potential. They lack domain expertise, to begin with; as a result of their primary training on public datasets, LLMs are naturally constrained in their capacity to respond in depth to domain-specific queries that fall outside the purview of their breadth of expertise. In the context of AI planning, attempts have been made to generate plans directly with LLMs. However, with GPT-4 achieving a mere 35% accuracy on simple planning problems, this method shows little promise [1]. Encouraging results in code translation have been shown by specialized LLMs like the StarCoder [2][3]. LLMs can perform zero-shot translation between languages after extensive pre-training on ultra-large code corpora, thereby making them a possible avenue for code translation. Aligned with this development, some research has been conducted in the translation of prompts in natural language into formal planning languages, including the Planning Domain Definition Language (PDDL) [4][5]. In accordance with the previous findings, translation to PDDL using LLMs was observed to be better than using them for the direct generation of plans.

The combined usage of natural language processing and formal planning languages offers a powerful synergy in automated planning systems. In this approach, LLMs serve as intermediaries that translate natural language instructions into structured PDDL representations. These PDDL outputs can then be fed into traditional symbolic planners, which have been refined over many years to ensure correctness of solutions and efficiency. This universalizes the concept of planning, making it accessible to people without modeling or planning expertise, while ensuring reliability of the plan thus formed.

That is not to say that this combination is infallible; it is a challenging task in itself to verify the correctness of code generation in LLMs. Some of the existing approaches leverage plan validators to assess whether the generated code can be solved by traditional planning algorithms [4], but this metric alone fails to capture the full scope of correctness. This is because the PDDL generated by the could be syntactically valid and solvable, yet entirely unrelated to the original prompt given by the user, creating a false sense of correctness.

To address these challenges, we build upon the Planetarium benchmark [6] by incorporating the DriverLog domain, enabling a focused evaluation of language models in translating natural language descriptions into PDDL for logistics-based planning tasks. Our approach goes beyond evaluating syntactic correctness by introducing semantic analysis through scene graphs, which capture the structural relationships in the initial and goal states of the planning problems. This allows for a meaningful comparison between the generated and ground truth PDDL files. We implement the edit distance metric that evaluates differences in graph edges, offering a detailed and accurate measure of semantic alignment. Through these improvements, we have attempted to provide a simple framework for assessing the capabilities of language models in structured planning tasks.

2 Background

To explain our approach, we begin by defining key concepts like the classical planning model and plan [7], PDDL [8][9], and scene graphs.

2.1 The Classical Planning Model

Classical planning involves selecting actions in environments that are deterministic and have a fully known initial state, that is, the states are fully observable. There is also a finite set of actions that can be performed on the states. The underlying model for classical planning can be represented as the following state model tuple:

$$\mathcal{S} = \langle S, s_0, S_G, A, f, c \rangle$$

where S represents a set of discrete and finite sets, $s_0 \in S$ is the initial state fully observed by the agent, $S_G \subseteq S$ is a non-empty set of goal states, A is a set of actions applicable $\forall s \in S$ such that $A(s) \subseteq A$, $f(a, s)$ is a deterministic transition function that leads to state s' on performing action $a \in A(s)$, and $c(a, s)$ is the positive cost associated with performing action a in state s.

2.2 Plan

The solution to a Classical Planning problem is a **plan**. Mathematically, it a sequence of applicable actions $a_0, a_1, ..., a_n$ that generate a sequence of states $s_0, s_1, ..., s_n, s_{n+1}$ such that s_{n+1} is a goal state.

2.3 Planning Domain Definition Language (PDDL)

PDDL is a formal language developed to standardize the representation of planning problems in AI. It caters to various planning paradigms, including classical planning problems. A PDDL planning problem is structured into a domain file and a problem file.

Static components of the planning environment, like the set of possible actions and the preconditions and effects that correspond with them, are defined in the domain file. Together, these components create rules that control state transitions in the planning problem. The problem file describes specific scenarios or instances of the overall environment by specifying its particular initial conditions and desired goals. Mathematically, it can be represented as the tuple:

$$\mathcal{P} = \langle D, I, G \rangle$$

where D is the domain, I is the initial state, i.e., a set of predicates that must hold at the start, and G is the goal state, i.e., a set of predicates that must hold at the end.

2.4 Syntactic vs. Semantic Correctness

Syntactic validation determines if the structure of a piece of content conforms to the rules of its corresponding formal language. It validates whether the generated or translated content complies with the syntax and grammar that the parser or system expects. This could be equivalent to verifying that parentheses are balanced, variables and constants are used consistently, etc. in the context of PDDL.

Semantic validation, on the other hand, extends beyond formal structure to ensure that the content makes sense in terms of its intended meaning. In the context of PDDL, this entails verifying that the relationships, actions, and goals stated in the problem are in line with the natural language prompt and that the resulting PDDL file appropriately represents the original planning problem.

2.5 Large Language Models (LLMs)

LLMs are deep learning models trained on massive text corpora to predict and generate text. They are based on transformer architectures, leveraging attention mechanisms to model dependencies

between words in a sequence. These models are parameterized by a large number of weights and are fine-tuned for various tasks.

Given an input sequence of tokens, $x = (x_1, x_2, ..., x_n)$ and a target sequence of tokens, $y = (y_1, y_2, ..., y_k)$, the LLM generates the conditional probability distribution for the next word in the sequence as:

$$P(y \mid x) = \prod_{i=1}^{k} P(y_i \mid x, y_1, \ldots, y_{i-1})$$

The model is trained to maximize the likelihood of this distribution.

2.6 Scene Graphs

A scene graph models the interactions between items and entities presenting an organized depiction of the environment or system. It is a directed graph with nodes representing items or things and edges representing the relationships or interactions between them. Scene graphs are used in planning activities to record the initial and goal states by characterizing the key elements involved as well as the functional relationships between them. These graphs are useful for capturing the semantic structure of a planning problem and with the interpretation and validation of generated PDDL representations.

A scene graph G can be represented as:

$$G = (V, E, R)$$

where V is a set of nodes representing the entities or objects in the scene, E is a set of directed edges representing the relationships between entities, and R is a set of edge labels that describe the specific types of relationships. The semantic structure of the scene graph is determined by the set of relationships in R.

2.7 Edit Distance

In the context of strings, **edit distance** measures the minimum number of operations (insertion, deletion, or substitution) needed to transform one string into another.

For graphs, Graph Edit Distance (GED) is a metric that quantifies the dissimilarity between two graphs. It measures the minimum number of edit operations required to transform one graph into another. These operations include node/edge insertion, node/edge deletion, and node/edge relabeling. The graph edit distance is the sum of the costs associated with these operations, where the goal is to find the minimum cost to transform one graph into another.

Our method modifies the traditional approach to graph edit distance by treating node relabeling as a combination of edge deletion and edge addition, allowing us to capture both semantic and structural changes in graph states. This distinction allows us to track changes in node and edge labels (such as objects or relationships being moved or transformed), which is more important in assessing the correctness of generated plans and their alignment with ground truth than the cost or optimization associated with the traditional GED.

3 Related Work

Dagan et. al. [10] created the LLM-Dynamic Planner framework that utilizes a neuro-symbolic approach, which uses LLMs to use instructions to get PDDL that is then fed to a symbolic planner. They merge the capability of LLMs to handle noise and uncertainity with the efficiency of symbolic planners. THeir biggest advantage is handling of uncertainity by generating acceptable predicates for unobserved objects, which is done through semantic and pragmatic inference. Silver et. al. [11] explore the planning capabilities of LLMs using a few-shot approach. They provide a set of problem and solution as few-shot examples, all in PDDL format, and make it generate the solution PDDL for a new problem that is presented. The responses are validated either by skipping syntactically invalid actions, or constrain actions to be of valid syntax through autoregressive prompting. Gestrin et. al. [12] use LLMs to incrementally extract information from natural language prompts, and subsequently generate PDDL code. They evaluate the plans with a classical planner, and utilize feedback from

either humans or LLMs to improve responses. Silver et. al. [13] test the planning as well as LLM parsing capabilities of GPT-4 by providing domain and problem PDDL files and making the LLM generate a python code that implements the plan for the problem. They additionally ask the LLM to summarize the domain before generating the python script to ensure that the LLM has understood the domain properly. Smirnov1 et. al. [14] attempted to improve the quality of PDDL planning of LLMS by using a generative approach that converts detailed natural language problem descriptions to an intermdiate JSON format. This JSON is updated based on feedback, and then subsequently converted to parseable PDDL.

Oswald et. al. [15] generate PDDL out of natural language descriptions generated from already defined PDDL, using in-context prompting. The text descriptions are generated using different strategies, starting with basic description and building up with alterations such as flipping and randomization. Steina et. al. [16] present the AutoPlanBench framework aimed at translating PDDL into natural language text by directly prompting in GPT-4. They also generate domain descriptions for the problem using action choices tried out by different mechanism involving various prompting techniques such as CoT, ReAct etc. Zhang et. al. [17] introduce the LAMMA-P, a multi-agent PDDL planner that combines the reasoning capabilities of LLMS with heuristic search techniques of planners. They also introduced MAT-THOR, a benchmark to evaluate multi-agent frameworks. Liu et. al. [18] address the inability of LLMs to solve long-horizon robot problems using a 3-step approach: natural language to PDDL conversion using in-context learning, plan evaluation using a classical planner, and finally conversion of plan back to natural language that is easily understood by humans. Guan et. al. [19] generate symbolic representations of actions in PDDL using natural language, which can be used to generate world models that can be executed in external domain-independent planners.

4 Methodology

The goal of this project is to evaluate the ability of LLMs to translate natural language descriptions of planning tasks into valid PDDL code. To rigorously assess the quality of the generated PDDL files, we go beyond traditional syntactic validation by implementing semantic evaluation. We employ scene graphs to represent the initial and goal states of PDDL problems and evaluate the structural alignment between generated and ground truth scene graphs using a modified edit distance.

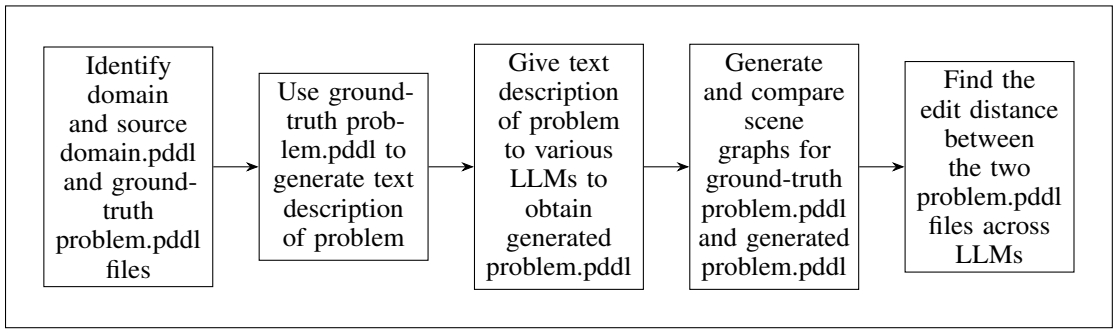


Figure 1: Project pipeline.

The methodology adopted by this project is heavily inspired by the work of Zuo et al. on Planetarium [6], where a rigorous benchmark was proposed for translating text to PDDL specifically for the BlocksWorld and Gripper domains. We have attempted to adapt their work to the Driver Logistics (DriverLog) domain. Some liberties have been taken in deviating from the base paper keeping in mind the resources available to us.

4.1 Dataset Preparation

The DriverLog planning domain was adopted from the 2002 International Planning Competition (IPC) domains [20]. It contains tasks associated with drivers, trucks, and their interactions with different locations. This domain imitates a logistics setting in which drivers must travel between locations and load, drive, and unload vehicles. The domain in PDDL specifies the types of objects that are characters in this scene, including drivers, trucks, and locations, as well as the predicates that

show the system's current state, including object locations and their relationships. The actions in this domain define the possible operations that can be performed within the logistics environment. The set of actions are as follows:

- **LOAD-TRUCK:** The truck and the object must be at the same location. The object is loaded into the truck, and it is no longer at the original location.
- **UNLOAD-TRUCK:** The truck and the object must be at the same location, and the object must be inside the truck. The object is removed from the truck and placed at the specified location.
- **BOARD-TRUCK:** The truck must be at the specified location, the driver must be at the same location, and the truck must be empty. The driver boards the truck, starting to drive it, and the truck is no longer empty.
- **DISEMBARK-TRUCK:** The truck must be at the specified location, and the driver must be driving the truck. The driver disembarks from the truck and arrives at the specified location, leaving the truck empty.
- **DRIVE-TRUCK:** The truck and driver must be at the initial location, and there must be a link between the two locations. The truck is moved from the starting location to the destination location.
- **WALK:** The driver must be at the starting location, and a path must exist between the two locations. The driver walks from the starting location to the destination location.

The problem files were categorized into various subsets of complexity; STRIPS, Numeric, Hard-Numeric, Simple-Time, and Time. We focus on classical planning problems within the STRIPS subset of PDDL. The problem files obtained from IPC serve as the ground-truth problem files which form the bases for getting the natural language descriptions of the problem.

`Llama-3.1-8B-Instruct` was used to generate two text description of each ground-truth problem.pddl file - an explicit description, where the instructions are apparent and technical, and an abstract description, where the instructions are comparatively vaguer. This involved extensive prompt engineering to ensure no additional text apart from an all-encompassing description was generated by the LLM.

These steps led to the formation of a dataset for the DriverLog domain which contained the ground-truth problem.pddl and its text description.

4.2 PDDL Generation and Syntactic Validation

We employed several LLMs, including Llama 3.1, GPT-4o, GPT-4o-mini, and Claude 3.5 (Haiku) to generate problem PDDL files from the provided natural language descriptions. These models were selected based on their relevance to code generation and general natural language processing tasks. For each problem instance, the same domain.pddl file was given as context along with the natural language description, and the generated PDDL files were collected for evaluation.

To ensure the basic validity of the generated PDDL files, we performed syntactic validation as an initial step. This process involved parsing the PDDL files using standard tools to verify whether they adhered to the structural and syntactical rules of the language. Since we considered the STRIPS set of problems, a model output is parseable if a PDDL parser supporting `:strips` can parse a valid PDDL problem from a substring in the output and if it can be converted into the scene graph representation discussed below. A problem is solvable if it is parseable and there exists a plan that can be applied to the initial scene that results in the goal scene. We validated whether the generated problems were solvable by using the BFS Planner, ensuring that they represented logically consistent problems.

4.3 Semantic Validation

Once the scene graphs are generated, we then computed the edit distances. For every problem, We computed two pairs of adjacency matrices: one for initial state graphs (generated and ground-truth) and one pair for goal state graphs (generated and ground-truth). We then computed the edit distance for both pairs of adjacency matrices, resulting in two edit distance values, one corresponding to the two initial state graphs and another corresponding to the two goal state graphs.

5 Results

We present a comparative analysis of Llama-3.1, Claude 3.5 (Haiku), GPT 4o, and GPT 4o-mini in this section. Figure 2 shows the solvability of the generated problem.pddl file for each model when explicit and abstract natural language problem descriptions are given to them. For reference, all prompting was zero shot.

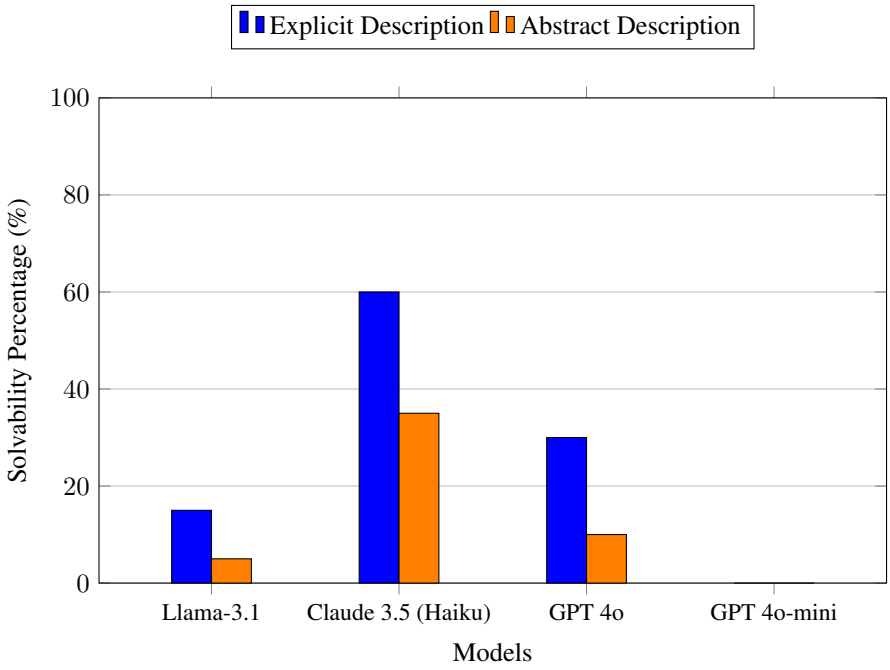
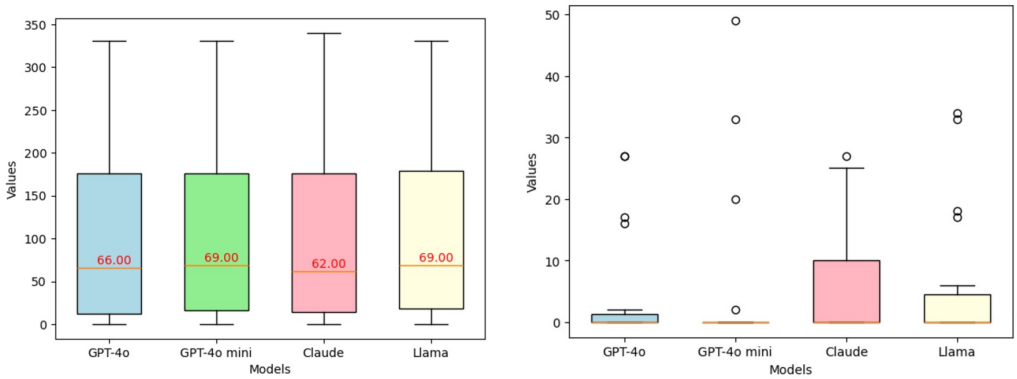


Figure 2: Solvability per model for explicit and abstract natural language descriptions of the problem.

GPT 4o-mini shows the worst solvability of the generated problem.pddl for both explicit and abstract problem descriptions, while Claude shows promising results for solvability, with 60% of its generated problem.pddl files being parseable and giving valid plans with the BFS-FF parser. Since the results of explicit problem descriptions are better, we have considered the problem.pddl files generated by means of explicit problem descriptions only.



(a) Edit distances for initial state scene graphs.

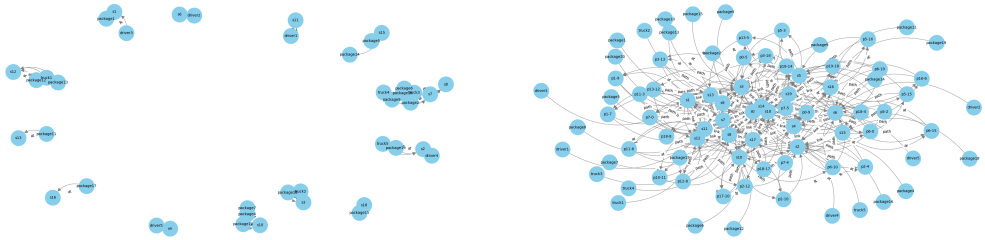
(b) Edit distances for goal state scene graphs.

Figure 3: Comparison of edit distances between generated and ground-truth problem files.

Figure 3(a) shows the median edit distance for the initial state of the generated problem file for each model with the ground-truth problem file. A very minimal difference is seen between the models. In Figure 3(b), which shows the edit distance for the goal state between the generated and ground-truth

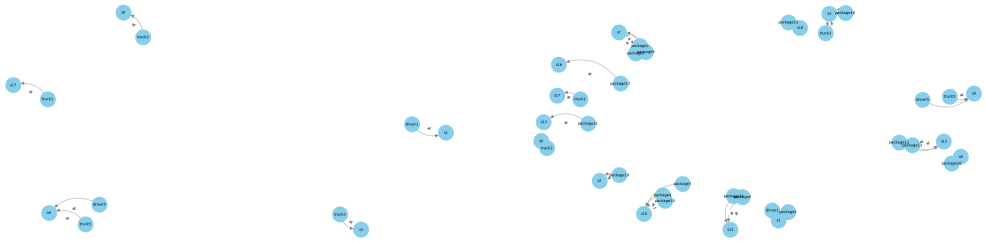
files, we see that while the median edit distance is 0 for all the models, Claude shows the highest maximum edit distance value.

Figures 4 and 5 visually represent the increased complexity of relationships between entities shown by the problem file generated by Claude.



(a) Ground-truth problem initial state. (b) Claude-generated problem initial state.

Figure 4: Initial state scene graphs.



(a) Ground-truth problem goal state. (b) Claude-generated problem goal state.

Figure 5: Goal state scene graphs.

A conclusion that can be drawn from these observations is that simply because Claude produces the most solvable, that is, the most syntactically-correct PDDL code, we cannot assume that it semantically matches what the problem actually intends to do.

6 Limitations and Future Scope

Metrics based on matching segments of generated code with the ground truth are limited in scope, as they fail to account for wide range of possible equivalent representations. This challenge stems from the fact that multiple PDDL representations can describe the same planning problem, yet identifying equivalence is often non-trivial. Our current metrics do not identify equivalent versions of the code. Moreover, this study is focused on the STRIPS subset of PDDL. Broadening the scope to include more expressive subsets, such as non-deterministic, temporal, or numeric domains, would provide a more robust and comprehensive evaluation.

References

[1] K. Valmeekam, M. Marquez, A. Olmo, S. Sreedharan, and S. Kambhampati, *Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change*, 2023. arXiv: `2206.10498 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2206.10498`.

[2] R. Li, L. B. Allal, Y. Zi, *et al.*, "Starcoder: May the source be with you!" *arXiv preprint arXiv:2305.06161*, 2023.

[3] R. Pan, A. R. Ibrahimzada, R. Krishna, *et al.*, "Lost in translation: A study of bugs introduced by large language models while translating code," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[4] L. Guan, K. Valmeekam, S. Sreedharan, and S. Kambhampati, "Leveraging pre-trained large language models to construct and utilize world models for model-based task planning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 79 081–79 094, 2023.

[5] B. Liu, Y. Jiang, X. Zhang, *et al.*, "Llm+ p: Empowering large language models with optimal planning proficiency," *arXiv preprint arXiv:2304.11477*, 2023.

[6] M. Zuo, F. P. Velez, X. Li, M. L. Littman, and S. H. Bach, "Planetarium: A rigorous benchmark for translating text to structured planning languages," *arXiv preprint arXiv:2407.03321*, 2024.

[7] H. Geffner and B. Bonet, *A concise introduction to models and methods for automated planning*. Morgan & Claypool Publishers, 2013.

[8] D. M. McDermott, "The 1998 ai planning systems competition," *AI magazine*, vol. 21, no. 2, pp. 35–35, 2000.

[9] A. Gerevini and D. Long, "Plan constraints and preferences in pddl3," Technical Report 2005-08-07, Department of Electronics for Automation ..., Tech. Rep., 2005.

[10] G. Dagan, F. Keller, and A. Lascarides, "Dynamic planning with a llm," *arXiv preprint arXiv:2308.06391*, 2023.

[11] T. Silver, V. Hariprasad, R. S. Shuttleworth, N. Kumar, T. Lozano-Pérez, and L. P. Kaelbling, "Pddl planning with pretrained large language models," in *NeurIPS 2022 foundation models for decision making workshop*, 2022.

[12] E. Gestrin, M. Kuhlmann, and J. Seipp, "Nl2plan: Robust llm-driven planning from minimal text descriptions," *arXiv preprint arXiv:2405.04215*, 2024.

[13] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. Kaelbling, and M. Katz, "Generalized planning in pddl domains with pretrained large language models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, 2024, pp. 20 256–20 264.

[14] P. Smirnov, F. Joublin, A. Ceravola, and M. Gienger, "Generating consistent pddl domains with large language models," *arXiv preprint arXiv:2404.07751*, 2024.

[15] J. Oswald, K. Srinivas, H. Kokel, J. Lee, M. Katz, and S. Sohrabi, "Large language models as planning domain generators," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 34, 2024, pp. 423–431.

[16] K. Steina, D. Fišera, J. Hoffmanna, and A. Kollera, "Automating the generation of prompts for llm-based action choice in pddl planning,"

[17] X. Zhang, H. Qin, F. Wang, Y. Dong, and J. Li, "Lamma-p: Generalizable multi-agent long-horizon task allocation and planning with lm-driven pddl planner," *arXiv preprint arXiv:2409.20560*, 2024.

[18] B. Liu, Y. Jiang, X. Zhang, *et al.*, "Llm+ p: Empowering large language models with optimal planning proficiency," *arXiv preprint arXiv:2304.11477*, 2023.

[19] L. Guan, K. Valmeekam, S. Sreedharan, and S. Kambhampati, "Leveraging pre-trained large language models to construct and utilize world models for model-based task planning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 79 081–79 094, 2023.

[20] D. Long and M. Fox, "The 3rd international planning competition: Results and analysis," *Journal of Artificial Intelligence Research*, vol. 20, pp. 1–59, Dec. 2003, ISSN: 1076-9757. DOI: 10.1613/jair.1240. [Online]. Available: http://dx.doi.org/10.1613/jair.1240.