DELTA: Decomposed Efficient Long-Term Robot Task Planning using Large Language Models

Yuchen Liu[1,2], Luigi Palmieri[1], Ilche Georgievski[2] and Marco Aiello[2]

***Abstract*— Recent advancements in Large Language Models (LLMs) have revolutionized many research fields. In robotics, the integration of common-sense knowledge from LLMs into downstream tasks has drastically advanced the field by unlocking unprecedented levels of context awareness. Despite their vast collection of knowledge, LLMs may generate infeasible plans due to hallucinations or missing domain information. To address these challenges and improve plan feasibility and computational efficiency, we introduce DELTA, a novel LLM-informed task planning approach. By using scene graphs as environment representations within LLMs, DELTA achieves rapid generation of precise planning problem descriptions. To enhance planning performance, DELTA decomposes long-term task goals with LLMs into an autoregressive sequence of sub-goals, enabling automated task planners to efficiently solve complex problems. In our extensive evaluation, we show that DELTA enables an efficient and fully automatic task planning pipeline, achieving higher planning success rates and significantly shorter planning times compared to the state of the art.**

I. INTRODUCTION

Various powerful Large Language Models (LLMs) have been developed in the past years that are capable of producing human-like texts, programming code, and service compositions etc. [1]–[5]. Nowadays with more robots cooperating with humans in industrial and household settings [6], [7], e.g., performing household tasks such as cleaning (Fig. 1), many researchers use LLMs for solving robot Task And Motion Planning (TAMP) problems [8]–[15]. While directly using pre-trained LLMs to generate action plans for the robots tends to result in extremely low success rates in generating executable plans and completing the goals [16], [17], most of them use LLMs to extract common-sense knowledge to improve the performance of classical automated task planning approaches with respect to plan correctness, executability, and feasibility [8], [11], [14]. Several approaches use LLMs to generate task specifications defined in formal language, e.g., the domain and problem files programmed in the Planning Domain Definition Language (PDDL) [18], that can be solved by the off-the-shelf TAMP algorithms [10], [19], [20]. However, previous TAMP approaches were cumbersome as they required vast manual knowledge engineering and input from human experts. On the other hand, none of the approaches above tackle long-term task planning problems, which are particularly difficult to solve with the growing problem complexity [14].

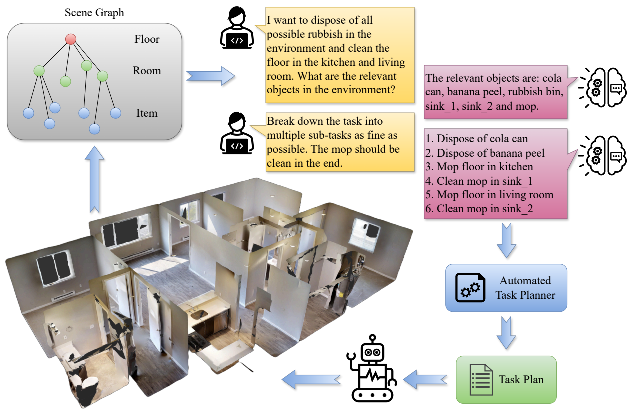
Scene Graph
Floor
Room
Item
I want to dispose of all possible rubbish in the environment and clean the floor in the kitchen and living room. What are the relevant objects in the environment?
Break down the task into multiple sub-tasks as fine as possible. The mop should be clean in the end.
The relevant objects are: cola can, banana peel, rubbish bin, sink_1, sink_2 and mop.
1. Dispose of cola can
2. Dispose of banana peel
3. Mop floor in kitchen
4. Clean mop in sink_1
5. Mop floor in living room
6. Clean mop in sink_2
Automated Task Planner
Task Plan

Fig. 1: An example of the task decomposition. A Scene Graph (SG) is pre-built from the environment [24]. Using the SG as the environment representation, a human user queries a LLM with goal descriptions to extract the relevant items and decompose the goal into multiple sub-goals. An automated task planner generates a task plan with respect to the sub-goals for the robot to execute.

Efficient environment representations are essential for robots solving long-term tasks in complex environments. Mapping mid-level perceptual data (e.g., 2D semantic segmentation) to high-level abstractions (e.g., environment topology) can be streamlined using Scene Graphs (SGs) [21]. Researchers have found that SGs provide compact spatial representations and enhance planning efficiency in task planning problems [13], [22], [23].

As it emerges from the state of the art, utilizing LLMs and automated task planning techniques to solve long-term robot task planning problems, with structured representations of large environments, still remains an open research topic. Therefore, we propose **DELTA**: ***D****ecomposed* ***E****fficient* ***L****ong-term* ***TA****sk planning for mobile robots using LLMs*, which is the first to fill the aforementioned vacancy. DELTA first feeds SGs into LLMs to generate the necessary domain and problem specifications in formal planning language, then decomposes the long-term task goals into multiple sub-ones using LLMs. The corresponding sub-problems are then solved autoregressively with an automated task planner. In summary, we present the following key contributions:

i) We introduce a novel combination of LLMs and SGs that enables the extraction of actionable and semantic knowledge from LLMs and its grounding into the environmental topology. Thanks to one-shot prompting, DELTA is capable of solving complex planning problems in unseen domains.

ii) We show that with the LLM-driven task decomposition strategy and the usage of formal planning language, compared to representative LLM-based baselines, DELTA is able to

[1]Corporate Research and Advance Engineering, Robert Bosch GmbH, Germany {yuchen.liu2, luigi.palmieri}@bosch.com
[2]Institute of Architecture of Application Systems, University of Stuttgart, Germany {yuchen.liu, ilche.georgievski, marco.aiello}@iaas.uni-stuttgart.de
This work was partly supported by the EU Horizon 2020 research and innovation program under grant agreement No. 101017274 (DARKO).

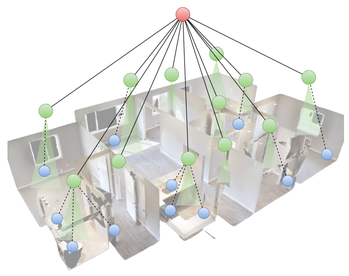

Fig. 2: *Shelbiana* scene [24] and the corresponding SG with floor, room, and item node layers. The edges refer to the semantic relationships. Not all item nodes are visualized.

complete long-term tasks with higher success rates, near-optimal plan quality, and significantly shorter planning time.

II. RELATED WORK

Despite LLMs' ability in temporal and few-shot reasoning [25], [26], they still struggle with analyzing complex spatial relationships and processing detailed environmental features [16]. Consequently, plans directly generated by LLMs are often not executable for the robots. Therefore, researchers have developed various approaches to ground LLM's output into executable and affordable action plans, or ground actionable knowledge into formal planning or programming language. Liu *et al.* [10] introduced *LLM+P* that translates NL problem descriptions into PDDL problem files with LLMs given user-provided PDDL domain files. Silver *et al.* [15] leveraged LLMs to comprehend the action knowledge from PDDL files and generate Python code to solve the problems. Despite the grounding with formal language, these approaches require handcrafted domain descriptions provided by human experts. They neither generalize to new domain knowledge, nor tackle long-term problems in large and complex environments.

Semantic understanding is a crucial factor for robot navigation in large environments. LLMs unlock the capability of reasoning over semantic relations embedded in large scenes and allow the capture of those relations from different scene representations, e.g., semantic maps [27], landmarks [28], [29], as well as SGs. Rana *et al.* [13] proposed *SayPlan* that uses LLMs to first conduct a semantic search through 3DSGs, then generates task plans upon the graph and refines iteratively, achieving grounded and scalable robot TAMP. But it still does not aim to solve long-term tasks.

However, since the probability of LLMs in producing incorrect output accumulates with growing planning horizon [16], most of the LLM-based approaches above have difficulties in tackling long-term planning problems. Thus, they mainly focus on semantically simple short-term tasks, e.g., object rearrangement, object-goal navigation, or other tasks that consist of a few such sub-tasks. The capability of LLMs to handle long-term tasks in large environments is not fully exploited. While decomposing a long-term task into multiple sub-tasks via classical machine learning methods can lead to a significant reduction of planning time [6], completing such a job with LLMs is still unexplored in the state-of-the-art.

III. METHODOLOGY

A. Problem Statement

We focus on solving long-term robot task planning problems with LLMs and consider mobile robot navigation tasks in household environments. The approach can also be generalized to other use cases. Given a SG as environment representation and domain and problem descriptions in NL, the LLM will generate the PDDL planning files and decompose the long-term goal into a sequence of sub-goals for solving the corresponding sub-problems autoregressively.

B. System Architecture

The architecture of DELTA is built around a five-step process (Fig. 3): domain generation, scene graph pruning, problem generation, goal decomposition, and autoregressive sub-task planning.

1) Domain Generation: The LLM takes an NL prompt describing the domain knowledge as input and generates a domain description file encoded in PDDL in a one-shot fashion. The prompt consists of three main parts: *role*, *example*, and *instruction*. The prompts in the following steps also have the same structure. After assigning a *role* description to the LLM, the necessary object types and the action knowledge, i.e., pre-conditions and effects, are given in the *example*. An instance of translating NL action description into PDDL is shown in Listing 1 in App. VII-A. Subsequently, the instruction depicts the action knowledge of a new domain. An example of the corresponding prompt design can be found in App. VII-B.

2) Scene Graph Pruning: Fig. 2 shows the hierarchical structure of a SG. The room nodes are annotated with their neighboring rooms, and the item nodes contain several attributes, e.g., accessibility, states, and affordable actions. In particularly large environments, SGs can contain a vast number of items, where not all items are relevant for accomplishing the given tasks. Therefore, we prompt the LLM to prune the SGs by only keeping the task-relevant items. An example of the prompt is listed in App. VII-B.

Pruning the SG allows a reduction of input tokens for the LLMs, thus reduce response time in generating the problem files. On the other hand, the more concise the information provided to the LLMs, the less likely that the LLMs generate erroneous output and hallucinations [16].

3) Problem Generation: In the generated problem file, the connections of rooms in the SG can be expressed using the *neighbor* predicate bi-directionally since the rooms are connected in both ways. The attributes of items can be defined with the predicates from the previously generated domain file, such as their positions and accessibilities. The LLM also translates the NL goal description into PDDL. An example of PDDL goal formulation is depicted in Listing 2 in App. VII-A. We again refer to App. VII-B for the prompt example.

4) Goal Decomposition: To improve the computational efficiency and reduce the complexity of the planning problem, we decompose the long-term goal defined in the problem file with LLMs. The prompt design is shown in App. VII-B.

Considering action preconditions and effects is essential in goal decomposition. E.g., action *mop_floor* requires

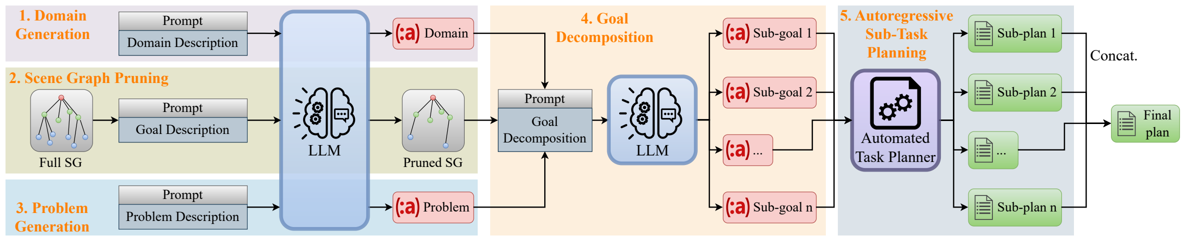

Fig. 3: The system architecture of DELTA with five steps: *Domain Generation*, *Scene Graph Pruning*, *Problem Generation*, *Goal Decomposition*, and *Autoregressive Sub-Task Planning*.

Algorithm 1: Autoregressive Sub-Task Planning

```
  Data: $\Pi$, $d$, $p_0$, $\boldsymbol{G}$
  Result: $\pi$
1 $\pi \leftarrow \varnothing$
2 extract $\boldsymbol{s}_0, \boldsymbol{g}_0$ from $p_0$
3 $\boldsymbol{s} \leftarrow \boldsymbol{s}_0$
4 for $\boldsymbol{g} \in \boldsymbol{G}$ do
5     $p \leftarrow$ replace $\boldsymbol{s}_0, \boldsymbol{g}_0$ in $p_0$ with $\boldsymbol{s}, \boldsymbol{g}$
6     $\pi', \boldsymbol{s}' \leftarrow \Pi(d, p)$
7     $\pi \leftarrow \text{concat}(\pi, \pi')$
8     $\boldsymbol{s} \leftarrow \boldsymbol{s}'$
9 end
```

mop_clean and results in *not (mop_clean)* as shown in Listing 1, it infers that one cannot mop the floor in another room continuously since the it turns dirty after mopping the previous one. Thus, the mop should be cleaned before going for the next room. An example of the corresponding decomposed goals is shown in App. VII-A.

5) Autoregressive Sub-Task Planning: As shown in Alg. 1, with the automated planner Π, the previously generated PDDL domain file d and problem file p_0, and the sequence of PDDL sub-goals $\boldsymbol{G}$ as inputs, since the initial states $\boldsymbol{s}_1$ of the first sub-problem p_1 are identical to the initial states $\boldsymbol{s}_0$ from the original problem p_0, thus, p_1 can be formulated by replacing the overall goal states $\boldsymbol{g}_0$ with the first sub-goal states $\boldsymbol{g}_1$.

The final states of each solvable sub-problem are exactly the initial states of the next sub-problem. Therefore, as shown in L. 4-9, after solving each sub-problem p, we obtain a sub-plan π' and the resulting final states $\boldsymbol{s}'$ (L. 6), which will then be assigned to $\boldsymbol{s}$ for the next sub-problem. The following sub-problems can be solved in the same way autoregressively. The final task plan π can be obtained by concatenating all sub-plans π' (L. 7), that only consist of executable actions.

IV. EVALUATION

In this section, we detail the metrics, domains, baselines, and datasets used for the evaluation.

A. Metrics

We evaluate the proposed system in terms of computational and task efficiency by the following metrics:

- **Success rate**: ratio of the succeeded trials to all trials. A trial is successful if the plan validator reports that the generated plan is valid, i.e., correct and executable.
- **Plan length**: number of actions in a plan. The decomposed plan length shows the length of the concatenated sub-plans from solving the sub-problems.

Models	PC	Dining	Cleaning	Office
LLM-As-Planner	70	38.67	0	0
LLM+P	76	4	0	0
LLM-GenPlan	88	80.67	3.33	0.67
SayPlan	68.67	70.67	54	40
DELTA (w/o dp.)	97.33	99.33	**80**	68.67
DELTA	**98**	**100**	**80**	**74.67**

TABLE I: Success rates [%] of each model. The results from each domain are averaged through all the scenes. *w/o dp.* refers to without goal decomposition.

- **Planning time**: inference time of the automated planner for finding a solution. The decomposed planning time refers to the total time of solving all sub-problems.

B. Evaluation Domains

We evaluate the approaches with five domains. The **Laundry** domain has a short-term task that serves as the example for one-shot prompting, where the robot is asked to bring the dirty clothes and detergent to the washing machine and then bring them to the bedroom after washed. Of the other domains, two have independent sub-tasks, namely, the **PC Assembly** domain (in the following abbreviated as *PC*) requires the robot to gather six different PC parts distributed in the environment and bring them together for assembly. In the **Dining Table Setup** domain (abbr. *Dining*), the robot should collect six different tableware from different rooms and place them on the dining table. Both domains can be decomposed into independent transportation sub-tasks. The remaining two domains have dependent sub-tasks which should be executed in a certain order. In the **House Cleaning** domain (abbr. *Cleaning*), the robot should first dispose of various rubbishes, then mop the floor in two rooms and clean the mop immediately after using it, and finally return to the hub for recharging. The **Home Office Setup** domain (abbr. *Office*) requires the robot to set up a home office by bringing four pieces of furniture, where some of them have contents inside that should be kept in the end, but they cannot be moved without unloading the contents. In each domain, the robot can only load one item at a time. Further details of the domains can be seen in App. VII-C.

C. Baselines

We select the following most popular and representative LLM-based task planning approaches as baselines:

LLM-As-Planner is a naive approach that directly queries the LLM to generate a high-level plan with a single prompt.

LLM+P [10] uses LLMs to translate NL problem descriptions into a PDDL problem file given user-provided PDDL domain files. It can be treated as a subset of DELTA with only

Metrics	Models	PC			Dining			Cleaning			Office		
		A	S	P	A	S	P	A	S	P	A	S	P
Plan Length	GT	**41**	**42**	**47**	**39**	**39**	**33**	**39**	**43**	**41**	**40**	**33**	**52**
	LLM-As-Planner	**41**	42.81	**47**	-	43	35	-	-	-	-	-	-
	LLM+P	**41**	**42**	**47**	-	**39**	-	-	-	-	-	-	-
	LLM-GenPlan	41.32	43.65	**47**	40.95	40.70	35	43.67	-	47	-	37	-
	SayPlan	44.45	48	47.24	41.83	46.97	35.48	45	48.11	43.29	46	42.85	56.47
	DELTA	**41**	**42**	**47**	**39**	**39**	35	40	44	45	**40**	**33**	**52**
Planning Time	DELTA	**0.0134**	**0.0144**	**0.0117**	**0.0101**	**0.0103**	**0.0089**	**0.0112**	**0.0120**	**0.0111**	**0.0170**	**0.0167**	**0.0149**
	DELTA (w/o dp.)	51.76	49.29	28.65	42.69	54.68	1.76	23.04	58.38	5.75	24.69	9.01	10.67

TABLE II: Further metrics of DELTA and other baselines of all domains and A(llensville), S(helbiana), and P(arole) scenes. The ground truth (*GT*) plan lengths are shown in the first row, indicating the optimal values. *"-"* means the metric is not applicable due to failures.

the problem generation step. In the following experiments, we provide a pre-defined PDDL domain file as input.

LLM-GenPlan [15] summarize the domain knowledge from given PDDL files, then propose a simple non-search-based strategy for solving the problem, and generates Python code to output a plan. The LLMs can refine the code using the error message from a plan validator with maximal 4 iterations.

SayPlan [13] first uses LLMs to determines a task-relevant sub-SG, then generate a high-level plan using the sub-SG and iteratively replans based on environmental feedback in at most 4 times. We implemented SayPlan on our own due to the lack of open-source code.

D. Dataset

We use four SGs from the 3D Scene Graph dataset [24]: *Kemblesville* is paired with *Laundry* domain as an example. *Allensville*, *Parole*, and *Shelbiana* are used for evaluations. The SGs are implemented as nested Python dictionaries.

E. Implementation and Parameters

We evaluate all approaches with pre-trained *GPT-4o* (version 2024-05-13) with default *temperature* and *top_p* parameters. We use Fast Downward (FD) [30] as the automated task planner with the default search configuration *seq-opt-lmcut* and the timeout of 60*s*, and PDDLGym [31] to obtain the world states, and VAL [32] for plan validation. Each experiment is repeated with 50 trials, resulting in 600 trials crosswise evaluated with 4 domains and 3 scenes.

V. RESULTS AND DISCUSSION

The evaluation results are displayed in Tables I and II. **LLM-As-Planner** performs the worst among all approaches, which proves that LLMs still have difficulties in discovering the underlying dependencies of complex tasks [17].

By grounding NL into PDDL, **LLM+P** achieves slightly higher success rates in the *PC* domain and it is also able to reach the optimal plan length. However, it only succeeded 4% in *Dining* and never in *Cleaning* and *Office* domains. The leading failure is planner timeout. Since LLM+P consumes the unpruned SGs, the complexity of the planning problem increases exponentially with the growing number of items, resulting in exceeding the planner's time limit.

LLM-GenPlan learns the domain knowledge encoded in PDDL and generalizes to solve unseen tasks. It achieves around 80% success rates and near-optimal plan lengths in the domains with independent sub-tasks (*PC* and *Dining*). Nonetheless, it mostly fails in the other more complex domains. Its capability to tackle more complicated problems is greatly limited by the usage of simple and non-search-based problem-solving strategies.

SayPlan achieves slightly lower success rates in *PC* and *Dining* domains than LLM-GenPlan, but considerably higher in the complex ones (54% in *Cleaning* and 40% in *Office*). Both approaches have replanning mechanisms, but LLM-GenPlan relies on the plan validator instead of environmental feedback, and therefore lacks affordance information such as wrong item location and room connections.

Finally, **DELTA** reaches the highest performance in all domains. The last two rows of Table I infer that the goal decomposition marginally improves its success rates since the original problems with undecomposed goals have significantly higher complexity, which occasionally leads to the planner timeout. Further leading failure cases are incorrectly generated predicates and missing attributes. The key factors that enable DELTA for long-term planning are grounding the actionable knowledge into formal planning language and relying on automated planners to find optimal solutions.

Goal decomposition also contributes to a significant reduction of the planning time by four orders of magnitudes, thus enabling a vast enhancement of planning efficiency. As shown in the lower part of Table II, the planning time is over $3,000$ times faster in *PC* and *Dining* domains, and almost $2,000$ time faster in *Cleaning* and *Office* domains on average.

VI. CONCLUSION

Suffering with LLMs' hallucinations and the extensive manual annotations of classical planning techniques, to address these challenges and improve plan feasibility and efficiency, we introduced DELTA, a novel LLM-informed task planning approach. DELTA's integration of scene graphs and LLMs facilitates the rapid generation of precise planning problem descriptions. To enhance planning performance, DELTA decomposes long-term task goals with LLMs into a sequence of sub-goals, enabling automated task planners to efficiently solve complex problems. In our evaluation, we show how DELTA enables a significant enhancement of efficiency in automated task planning in terms of a considerably faster planning time and higher success rates compared to various baselines. For future work, we plan to implement repairing mechanisms for handling uncertainties in dynamic environments, ablate our approach with more LLMs, and validate on real-world robot operations.

VII. APPENDIX

A. Translating Natural Language into PDDL Formulations

1) Action Formulation: For instance, the "*mop_floor*" action can be described as "*For mopping the floor, the agent is in the room and has the mop in hand, the mop is clean while the floor is not clean. After the action, the floor is clean, but the mop is not clean anymore, and the agent's battery will no longer be full.*" The corresponding action can be formulated in PDDL as follows:

Listing 1: Action "*mop floor*" defined in PDDL

```
(:action mop_floor
    :parameters (?a - agent ?i - item ?r - room)
    :precondition (and
        (agent_at ?a ?r)
        (item_is_mop ?i)
        (item_pickable ?i)
        (agent_has_item ?a ?i)
        (mop_clean ?i)
        (not(floor_clean ?r))
    )
    :effect (and
        (floor_clean ?r)
        (not(mop_clean ?i))
        (not(battery_full ?a))
    )
)
```

2) Goal Formulation: The goal of a house cleaning problem shown in Fig. 1 given by the human user can be formulated as follows:

Listing 2: Goal states of the house cleaning problem in PDDL

```
(:goal
    (and
        (item_disposed cola_can)
        (item_disposed banana_peel)
        (floor_clean living_room)
        (floor_clean kitchen)
        (mop_clean mop)
    )
)
```

3) Goal Decomposition: As indicated in the *mop_floor* action, it requires the predicate *mop_clean* as precondition, and results in the state *not (mop_clean)*. It infers that the mop turns dirty after cleaning a room, and therefore cannot be used to directly mop the next room. A cleaning action is thus necessary. As such, the goal states in Listing 2 can be decomposed as follows:

Listing 3: Decomposed goals of the house cleaning problem

```
(:goal (item_disposed cola_can))
(:goal (item_disposed banana_peel))
(:goal (floor_clean living_room))
(:goal (mop_clean mop))
(:goal (floor_clean kitchen))
(:goal (mop_clean mop))
```

B. Prompt Design

The prompt structures for domain generation, SG pruning, problem generation, and long-term goal decomposition are formulated as follows (the *purple* and *blue* text refers to NL description and programming code, respectively):

Prompt Structure for Domain Generation

Role: You are an excellent domain generator. Given a description of domain knowledge, you can generate a PDDL domain file.
Example: A robot in a household environment can perform the following *example object types* and *example actions with preconditions and effects*. The corresponding action definitions in a PDDL domain file look like: *example_domain.pddl*.
Instruction: A new domain has the following *new object types and actions*. Please generate a corresponding PDDL domain file.

Prompt Structure for Scene Graph Pruning

Role: You are an excellent assistant in pruning SGs with a list of SG items and a goal description.
Example: A SG can be programmed as a nested Python dictionary such as *example_sg.py*. For accomplishing the *example goal*, the relevant items are *[example_relevant_items]*.
Instruction: Given a new *query_sg.py* and a new *goal description*, please prune the SG by keeping the relevant items.

Prompt Structure for Problem Generation

Role: You are an excellent problem generator. Given a SG and desired goals, you can generate a PDDL problem file.
Example: Given an *example_sg.py*, an *example goal description*, and using the predicates defined in *example_domain.pddl*, a corresponding PDDL problem file looks like: *example_problem.pddl*.
Instruction: Given a new *query_sg.py*, a new *goal description*, please generate a new PDDL problem file using the predicates in the previously generated *query_domain.pddl*.

Prompt Structure for Goal Decomposition

Role: You are an excellent assistant in decomposing long-term goals. Given a PDDL problem file, you can decompose the goal states into a sequence of sub-goals.
Example: Given an *example_problem.pddl*, the goal states can be decomposed into a sequence of *example sub-goals*. Using the predicates defined in *example_domain.pddl*, the *example sub-goals* can be formulated as: *sub-goal_1.pddl*, ..., *sub-goal_n.pddl*.
Instruction: Given the *query_problem.pddl* generated previously, please decompose the goal considering the predicates and actions from the previously generated *query_domain.pddl*.

C. Evaluation Domains

1) Laundry Domain: New action: *For laundering, you need the clothes, the detergent, and the washing machine, where the clothes and detergent are pickable, and the washing machine can be turned on. The agent and all items should be in the same room and the agent is not loaded. As result, clothes will be clean.* Goal: *Launder the clothes and bring them to bedroom_1.*

2) PC Assembly Domain: New action: *For assembling a pc, you need a mainboard, a CPU (Central Processing Unit), a RAM (Random-Access Memory), a SSD (Solid-State-Drive), a GPU (Graphics Processing Unit), and a PSU (Power Supply Unit), where all items are pickable. The agent and all items should be in the same room, agent isn't loaded. As result, the PC is assembled.* Goal: *Bring the necessary PC parts to the living room and assemble the PC.*

3) Dining Table Setup Domain: New action: *For placing item_1 on item_2, the agent and item_2 must be in the same room, item_1 is be pickable and the agent has it in hand. As result, item_1 is on item_2, the agent is not loaded anymore.* Goal: *Set up the dining table for dinner, place the tableware (i.e., a plate, a fork, a knife, a spoon, and a glass) on the* *dining table. Also bring something romantic (i.e., flower) to the dining table.*

4) House Cleaning Domain: New actions:

- *For disposing of an item, you need a rubbish bin, the agent should be in the same room with the rubbish bin, and is loaded with the pickable item to be disposed of. As result, the item will be disposed of, the agent is not loaded anymore, and battery will not be full.*
- *Mop floor*, see Listing 1
- *For cleaning mop, the agent must have a pickable and unclean mop in hand, and should be in the same room with a sink. As result, mop will be clean and lies in the room, agent is not loaded, and battery will not be full.*
- *For charging the agent, it should be in the same room with the robot hub, the agent is not loaded, and the hub is accessible. As result, agent's battery will be full.*

Goal: *Identify and dispose of the possible rubbish (e.g. food residue, drink bottles/cans etc.), mop the floor in living room and kitchen. The mop should be clean in the end, and the battery should be full.*

5) Home Office Setup Domain: New actions:

- *For loading item_2 into item_1, the agent and item_1 must be in the same room, item_1 must be loadable and empty, item_2 must be pickable and is loaded by the agent. As result, item_2 will be in item_1, and the agent is not loaded anymore.*
- *For unloading item_2 from item_1, the agent and item_1 must be in the same room, item_1 must be loadable and not empty, item_2 must be pickable and lie in item_1, the agent is not loaded. As result, item_2 is not in item_1 anymore, and item_1 becomes empty.*

Goal: *Set up a home office in the living room by bringing the desk, lamp, locker and shelf along with the contents inside. Note that you cannot move a loadable item if it is not empty.*

REFERENCES

[1] J. Achiam *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] A. Chowdhery *et al.*, "Palm: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[4] H. Touvron *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[5] M. Aiello and I. Georgievski, "Service composition in the chatgpt era," *Service Oriented Computing and Applications*, pp. 1–6, 2023.

[6] Y. Liu, L. Palmieri, I. Georgievski, and M. Aiello, "Human-flow-aware long-term mobile robot task planning based on hierarchical reinforcement learning," *IEEE Robotics and Automation Letters*, 2023.

[7] N. Aboki, I. Georgievski, and M. Aiello, "Automating a telepresence robot for human detection, tracking, and following," in *Annual Conference Towards Autonomous Robotic Systems*. Springer, 2023.

[8] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9118–9147.

[9] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, "Llm-planner: Few-shot grounded planning for embodied agents with large language models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 2998–3009.

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

[11] Y. Ding, X. Zhang, C. Paxton, and S. Zhang, "Task and motion planning with large language models for object rearrangement," *arXiv preprint arXiv:2303.06247*, 2023.

[12] M. Ahn *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[13] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf, "Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning," in *Conference on Robot Learning*. PMLR, 2023, pp. 23–72.

[14] Y. Chen, J. Arkin, Y. Zhang, N. Roy, and C. Fan, "Autotamp: Autoregressive task and motion planning with llms as translators and checkers," *arXiv preprint arXiv:2306.06531*, 2023.

[15] 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, no. 18, 2024, pp. 20 256–20 264.

[16] V. Pallagani *et al.*, "On the prospects of incorporating large language models (llms) in automated planning and scheduling (aps)," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 34, 2024, pp. 432–444.

[17] K. Valmeekam, A. Olmo, S. Sreedharan, and S. Kambhampati, "Large language models still can't plan (a benchmark for llms on planning and reasoning about change)," *arXiv preprint arXiv:2206.10498*, 2022.

[18] D. McDermott *et al.*, "Pddl-the planning domain definition language," 1998.

[19] Y. Xie, C. Yu, T. Zhu, J. Bai, Z. Gong, and H. Soh, "Translating natural language to planning goals with large-language models," *arXiv preprint arXiv:2302.05128*, 2023.

[20] 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.

[21] Z. Ravichandran, L. Peng, N. Hughes, J. D. Griffith, and L. Carlone, "Hierarchical representations and explicit memory: Learning effective navigation policies on 3d scene graphs using graph neural networks," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 9272–9279.

[22] C. Agia *et al.*, "Taskography: Evaluating robot task planning over large 3d scene graphs," in *Conference on Robot Learning*. PMLR, 2022.

[23] G. Chalvatzaki, A. Younes, D. Nandha, A. T. Le, L. F. Ribeiro, and I. Gurevych, "Learning to reason over scene graphs: a case study of finetuning gpt-2 into a robot language model for grounded task planning," *Frontiers in Robotics and AI*, vol. 10, 2023.

[24] I. Armeni *et al.*, "3d scene graph: A structure for unified semantics, 3d space, and camera," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[25] T. Brown *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[26] S. Xiong, A. Payani, R. Kompella, and F. Fekri, "Large language models can learn temporal reasoning," *arXiv preprint arXiv:2401.06853*, 2024.

[27] B. Chen *et al.*, "Open-vocabulary queryable scene representations for real world planning," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 11 509–11 522.

[28] D. Shah, B. Osiński, S. Levine *et al.*, "Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action," in *Conference on Robot Learning*. PMLR, 2023, pp. 492–504.

[29] S. Wang *et al.*, "Less is more: Generating grounded navigation instructions from landmarks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 15 428–15 438.

[30] M. Helmert, "The Fast Downward Planning System," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[31] T. Silver and R. Chitnis, "Pddlgym: Gym environments from pddl problems," in *International Conference on Automated Planning and Scheduling (ICAPS) PRL Workshop*, 2020.

[32] R. Howey, D. Long, and M. Fox, "VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL," in *16th IEEE International Conference on Tools with Artificial Intelligence*. IEEE, 2004, pp. 294–301.