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

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

***Abstract*— Recent advancements in Large Language Models (LLMs) have sparked a revolution across many research fields. In robotics, the integration of common-sense knowledge from LLMs into task and motion planning has drastically advanced the field by unlocking unprecedented levels of context awareness. Despite their vast collection of knowledge, large language models 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. Project webpage: `https://delta-llm.github.io/`**

I. INTRODUCTION

With the rapid and enormous progress in the research field of Natural Language Processing (NLP), various powerful Large Language Models (LLMs) have been developed that are capable of producing human-like texts, programming code, and service compositions etc. [1]–[5]. Nowadays with more and 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, inducing practically impossible domain adaptations. 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].

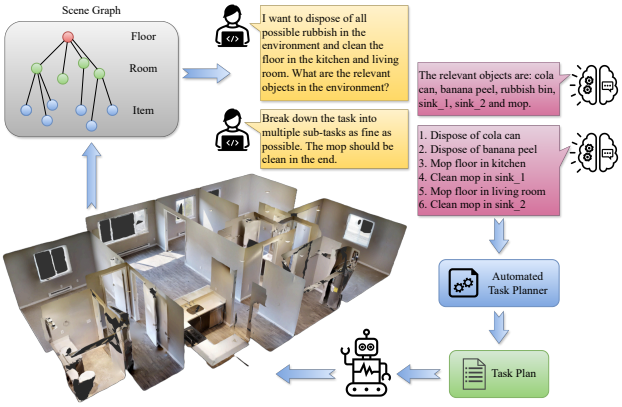

Fig. 1: An example of the long-term task decomposition. A Scene Graph (SG) is pre-built from the environment [25]. Using the SG as the environment representation, a human user queries a LLM with goal descriptions to extract the relevant items and decompose the goal into multiple sub-goals. An automated task planner generates a task plan with respect to the sub-goals for the robot to execute.

For robots solving long-term task sequences in large and complex environments, having efficient environment representations is a crucial prerequisite for the robot to understand the semantic information [21]. While mapping the mid-level perceptual representations (e.g., 2D semantic segmentation) into more appropriate high-level abstractions (e.g., environment topology and semantic relations between objects) can be costly and complex [22], it can be solved efficiently using high-level representations such as Scene Graphs (SGs) [23]. For tackling task planning problems in such environments, researchers have found that SGs can serve as compact and informative spatial representations and can improve planning efficiency [13], [22], [24].

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 the best of our knowledge, 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

[1]Corporate Research and Advance Engineering, Robert Bosch GmbH, Germany {yuchen.liu2, luigi.palmieri, sebastian.koch2}@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).

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

II. RELATED WORK

A. 3D Scene Graphs

A 3D Scene Graph (3DSG) is a recent 3D scene representation used to model large real-world environments as a graph structure. They were first introduced by Armeni *et al*. [25] as a hierarchical model to connect buildings, rooms, objects, and humans in multiple layers. Following this introduction, Rosinol *et al*. [26], [27] and Hughes *et al*. [28] investigated the construction of 3DSGs from sensor data. While other approaches focus on modeling semantic relationships between objects [29]–[33] from 3D point clouds. With the rise of LLMs, 3DSGs also become open-vocabulary [34]–[36], offering a deeper understanding of relationships between objects. Recently, 3DSGs have started to be integrated into robotics systems. Applications include navigation [34], [35], [37] as well as task and motion planning [13], [24].

B. LLM-based Robot Task and Motion Planning

The rich embedded semantic and common-sense knowledge allows LLMs to proficiently understand Natural Language (NL) instructions and perform temporal reasoning, first-order logic translation, and few-shot or even zero-shot planning [38]–[40]. However, LLMs 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. Ahn *et al*. [12] proposed *SayCan* to constrain the LLMs with pre-trained skills when generating actions. 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 domain knowledge and problem specifications from PDDL files, then used LLMs to generate Python code to solve the problems. However, most of these approaches require handcrafted domain descriptions provided by human experts, and do not generalize to new domain knowledge. Moreover, they do not tackle long-term planning 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 [41], landmarks [42], [43], 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.

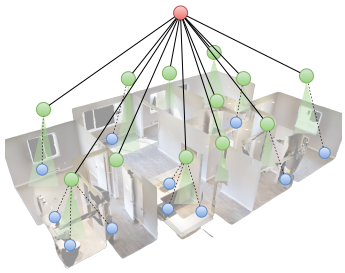

Fig. 2: *Shelbiana* scene [25] and the corresponding SG with floor, room, and item node layers. The edges refer to the semantic relationships. Not all item nodes are visualized.

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.

To distinguish the *object* keyword in PDDL from objects in SGs, in the following, we refer to the objects in SGs as *items*. We define *agent* as an object type in the domain file and *robot* as an instance of *agent* in the problem file. Furthermore, we assume full observability of all nodes in the SGs.

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 formal planning language, e.g., PDDL, correspondingly in a one-shot fashion.

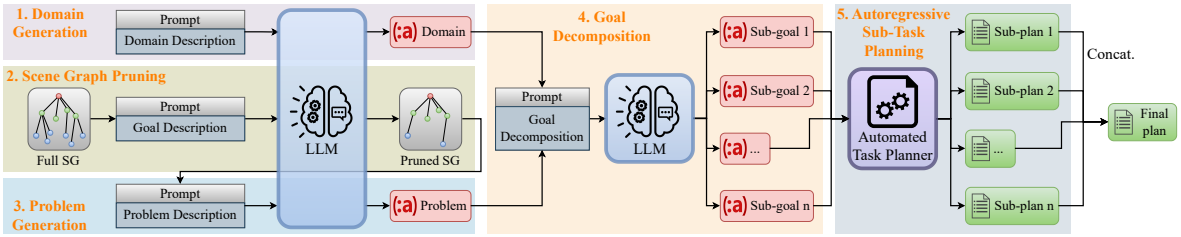

Fig. 3: The system architecture of DELTA with five steps: *Domain Generation*, *Scene Graph Pruning*, *Problem Generation*, *Goal Decomposition*, and *Autoregressive Sub-Task Planning*.

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))
    )
)
```

The prompt consists of three main parts: role, example, and instruction. The further prompts in the following steps also have the same structure. The example of the domain description includes the necessary object types and the action knowledge, i.e., pre-conditions and effects. 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 shown in Listing 1.

Subsequently, the instruction introduces the requirements for generating a new domain file. An overview of the prompt structure is shown as follows (the *purple* and *blue* text refers to NL description and programming code, respectively):

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 pre-conditions 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.

2) Scene Graph Pruning: SG has a hierarchical structure, An example of the layout is shown in Fig. 2. 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 large number of items, where not all items are relevant for accomplishing different tasks. Therefore, we prune the SGs with LLMs in the following way:

Role: You are an excellent assistant in pruning SGs with a list of SG items and a goal description.

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)
    )
)
```

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.

Pruning the SG allows a reduction of input tokens for the LLMs, thus achieving a fast 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: The prompt of this step can be similarly formulated in the following way:

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*.

In the generated problem file, the connections of rooms in the SG can be expressed using the *neighbor* predicate. For instance, if *kitchen* is connected with *corridor*, the relationship can be formulated as *(neighbor kitchen corridor)* and *(neighbor corridor kitchen)* since the rooms are connected bi-directionally. Similarly, 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. E.g., the goal of a house cleaning problem given by the human user can be formulated as shown in Listing 2, Fig. 1.

4) Goal Decomposition: To improve the computational efficiency and reduce the complexity of the planning problem, the long-term goal defined in the problem file can be decomposed with LLMs using the following prompt:

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*.

Taking action knowledge into account in goal decomposition is essential. For instance, knowing that the action *mop_floor* requires *mop_clean* and results in *not (mop_clean)* as shown in Listing 1, which infers that one cannot mop the floor in another room continuously since the mop turns dirty after mopping the previous room. Thus, cleaning the mop before mopping the next room should be considered when decomposing the goal, as shown in Fig. 1.

5) Autoregressive Sub-Task Planning: We use an automated task planner to solve the corresponding sub-problems one after another as shown in Algorithm 1. It takes the planner Π, the previously generated PDDL domain file d and problem file p_0 with undecomposed goals, and the sequence of PDDL sub-goals $\boldsymbol{G}$ as inputs. 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 simply formulated by replacing the undecomposed goal states $\boldsymbol{g}_0$ from p_0 with the first sub-goal states $\boldsymbol{g}_1$ from $\boldsymbol{G}$.

The final states resulting from the plan's execution of each solvable sub-problem are, in fact, exactly the initial states of the next sub-problem. Therefore, as shown in the for-loop 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. The success rates of each domain are averaged through the experiments with all three scenes.
- **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.
- **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.
- **Number of expanded nodes**: nodes in a search tree created and explored during the search process for solving a planning problem. A lower number implies lower problem complexity reported by the planner.

Algorithm 1: Autoregressive Sub-Task Planning

```
Data: Π, d, p_0, G
Result: π
1 π ← ∅
2 extract s_0, g_0 from p_0
3 s ← s_0
4 for g ∈ G do
5 |   p ← replace s_0, g_0 in p_0 with s, g
6 |   π', s' ← Π(d, p)
7 |   π ← concat(π, π')
8 |   s ← s'
9 end
```

B. Evaluation Domains

We run the evaluation on 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, i.e., a mainboard, a CPU, a GPU, a RAM, a SSD, and a Power Supply Unit (PSU), distributed in the environment and bring them to the living room for assembly. In the **Dining Table Setup** domain (abbr. *Dining*), the robot should collect a plate, a fork, a knife, a spoon, and a glass from different rooms, and also find something romantic, then 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 a cola can, a banana peel, and a rotting apple in the rubbish bin, then mop the floor in the kitchen and living room and clean the mop immediately after cleaning each room. Finally, the robot should return to the hub for recharging. The **Home Office Setup** domain (abbr. *Office*) requires the robot to set up a home office in the living room by bringing a desk, a lamp, a shelf, and a locker. The shelf and locker 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.

C. Baselines

We select four most popular and representative LLM-based task planning approaches: LLM-As-Planner, LLM+P [10], LLM-GenPlan [15], and SayPlan [13] according to the usage of formal language and environmental representation, as well as the capability of tackling long-term tasks shown in Table I.

LLM-As-Planner is a naive approach that directly queries the LLM to generate a high-level plan using a prompt that comprises all information, i.e., domain knowledge, environment and goal descriptions.

LLM+P [10] uses LLMs to translate NL problem descriptions into a PDDL problem file given user-provided PDDL domain files, for an automated task planner to solve the problem. It can be treated as a subset of DELTA with only the problem generation step. In the following experiments, we provide a pre-defined PDDL domain file as input.

Models	Formal planning language	Structured env. repr.	Long-term tasks
LLM-As-Planner	✗	✗	✗
LLM+P [10]	✓	✗	✓
LLM-GenPlan [15]	✓	✗	✗
SayPlan [13]	✗	✓	✗
DELTA (ours)	✓	✓	✓

TABLE I: Capabilities of different LLM-based models

LLM-GenPlan [15] uses LLMs to first summarize the domain knowledge from input PDDL files, then propose a simple generalized planning strategy without using search, and finally generates Python code that outputs a task plan. The LLMs can refine the code using the debug information from a plan validator with maximal 4 iterations.

SayPlan [13] first determines a task-relevant sub-SG with the LLM-based semantic search, then uses LLMs to generate a high-level plan using the sub-SG and iteratively replans based on environmental feedback. We implemented SayPlan on our own due to the lack of available open-source code. The number of maximal replanning iterations is set to 4.

D. Dataset

We use four scene graphs from the 3D Scene Graph dataset [25]: *Kemblesville* (9 rooms, 16 items) is paired with the *Laundry* domain. *Allensville* (11 rooms and 42 items), *Parole* (7 rooms, 31 items), and *Shelbiana* (12 rooms, 34 items) are used to evaluate the rest four domains. We implement the SGs as nested dictionaries in Python.

E. Implementation and Parameters

We evaluate DELTA with pre-trained *GPT-4-turbo* (version 2024-04-09), *GPT-4o* (version 2024-05-13), and *Llama-3.1-70B* with default *temperature* and *top_p* parameters. The other baselines are evaluated with *GPT-4o*.

We use Fast Downward (FD) [44] automated task planner with the default search configuration *seq-opt-lmcut* and the timeout of 60*s*. Moreover, we use PDDLGym [45] to obtain the world states, and the plan validation tool VAL [46] to validate the correctness and executability of the generated plans. Each experiment is repeated with 50 trials, resulting in 600 trials crosswise evaluated with 4 domains and 3 scenes in total. All approaches are implemented in Python 3.8.13. We run the experiments on a standard PC with an Intel Xeon W CPU at 3.40 GHz and 32 GB RAM. The *GPT* models are deployed with Azure OpenAI Service, while the *Llama* model with two Nvidia A100 GPUs in 4-bit quantization.

V. RESULTS AND DISCUSSION

The evaluation results are displayed in Tables II and III. **LLM-As-Planner** performs the worst among all approaches with 70% and 38.67% in domains with independent sub-tasks, and no successful case in those with dependent sub-tasks. Despite the ability in temporal reasoning, LLMs still have difficulties in discovering the underlying dependencies and preconditions of complex long-term tasks [17], [38].

By grounding NL into formal planning language, **LLM+P** is able to achieve slightly higher success rates in the *PC* domain and it is also able to reach the optimal plan length and number of expanded nodes. However, it only reaches 4% success rate in *Dining* and never succeeds in *Cleaning* and *Office* domains. The leading failure is planner timeout. Since LLM+P consumes the original SGs with a large number of items, although the LLMs have mostly transferred the items from SGs to the PDDL problem files, the complexity of the planning problem increases exponentially with the growing number of items, resulting in exceeding the planner's timeout.

Models	PC	Dining	Cleaning	Office
LLM-As-Planner	70	38.67	0	0
LLM+P	76	4	0	0
LLM-GenPlan (w/o rp.)	36.67	38.67	0	0
LLM-GenPlan	88	80.67	3.33	0.67
SayPlan (w/o rp.)	8.67	1.33	0	0.67
SayPlan	68.67	70.67	54	40
DELTA (Llama-3.1-70B)	34.67	23.33	0	0.67
DELTA (GPT-4-turbo)	93.33	74	32.67	9.33
DELTA (GPT-4o, w/o dp.)	97.33	99.33	**80**	68.67
DELTA (GPT-4o)	**98**	**100**	**80**	**74.67**

TABLE II: Success rates [%] of different models. The results from each domain are averaged through all the scenes. The upper part of the table shows the results of the baselines based on *GPT-4o*. The lower part lists the outcome of DELTA. *w/o rp.* and *w/o dp.* refer to without replanning and goal decomposition, respectively.

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. Since LLM-GenPlan solely utilizes LLMs to propose simple and non-search-based problem-solving strategies, its capability to tackle more complicated problems, i.e., problems with underlying preconditioned sub-ones, is greatly limited.

SayPlan achieves slightly lower success rates in the domains with independent sub-tasks than LLM-GenPlan, but considerably higher success rates in those with dependent sub-tasks (54% in *Cleaning* and 40% in *Office*), where LLM-GenPlan barely succeeded. Both approaches have replanning mechanisms, but LLM-GenPlan only relies on the plan validator, which checks invalid actions with unsatisfied preconditions. SayPlan on the other hand, obtains feedback from the SG simulator, which additionally provides environmental information when an action fails, e.g., cannot go to an unconnected room, wrong item location, or unaffordable action upon an item. By grounding the actionable knowledge into environmental topology (i.e., SGs), SayPlan is able to tackle more complex tasks then LLM-GenPlan.

Although replanning significantly improves the success rates of LLM-GenPlan and SayPlan, as indicated by their results with *(w/o replan)* in Table II, it merely ensures the executability of the generated plans but not the correctness and optimality, i.e., the plan reaches the goal with the shortest path. This is because LLMs generate output by predicting a probability distribution over the possible next tokens [16], instead of being a search- or an optimization-based planning process. The higher numbers of plan lengths of LLM-GenPlan and SayPlan in Table III further prove the statement. Thus, LLM-GenPlan and SayPlan are not suitable for long-term task planning, as shown in Table I.

Finally, **DELTA** achieved the highest success rates in all domains. The last two rows of Table II infer that the

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
Expanded Nodes	DELTA	**624.83**	**727**	**576**	**571.83**	**597.85**	**561.11**	**407**	**364**	**365.12**	**405.80**	**423.03**	**442.17**
	DELTA (w/o dp.)	1,585,185	1,317,615	1,379,036	1,016,429	2,187,261	368,728	1,561,834	1,797,434	107,400	501,148	149,024	321,221

TABLE III: Further metrics of DELTA with and without goal decomposition (*w/o dp.*) and other baselines in 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. All results are based on *GPT-4o* and are averaged over the succeeded cases.

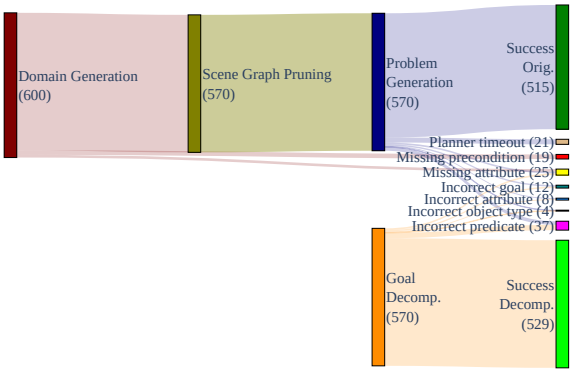

Fig. 4: Failure analysis of DELTA with *GPT-4o*. Each step, success, and failure type is annotated with the number of trials. *Problem Generation* and *Goal Decomposition* are decoupled since the planning of original and decomposed problems are independent and executed parallelly with the same number of trials outgoing from *Scene Graph Pruning* step.

goal decomposition marginally improves its success rates since the original problems with undecomposed goals have significantly higher complexity, as reflected by the number of expanded nodes in the lower part of Table III, which occasionally leads to the planner timeout. Fig. 4 indicates that 21 out of the 600 trials failed due to planner timeout. Further leading causes of failures are incorrectly generated predicates (37 out of 600 trials, such as the *neighbor* relationship of two unconnected rooms), missing attributes (25 out of 600 trials, such as *item_accessible*), etc.

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. As indicated in Table I and Sec. IV-C, with LLM+P being a subset of DELTA focusing solely on problem generation, and both DELTA and LLM+P translating NL into formal language, LLM+P can also handle long-term planning problems thanks to the usage of PDDL. However, LLM+P succeeded notably less than DELTA, because apart from goal decomposition, LLM+P also suffers from redundant scene representations, i.e., unpruned SGs, that prevent the automated planner from finding solutions efficiently. Having more sparse SGs notably reduces the risk of LLMs producing unnecessary and possibly incorrect predicates translated from the irrelevant items, which results in more potential planning errors.

Decomposing the long-term goal also contributes to a significant reduction of the planning time and the number of expanded nodes by four orders of magnitudes, thus enabling a vast enhancement of planning efficiency. As shown in the lower part of Table III, 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. The number of expanded nodes is also reduced by, on average, circa $2,000$ times in all the experiments.

Ablation: Besides *GPT-4o*, we further evaluate DELTA with *GPT-4-turbo* and *Llama-3.1-70B*. The corresponding success rates are listed in the lower part of Table II. Switching to *GPT-4-turbo* results in a notable performance drop, especially in the *Office* domain, where the success rate decreased significantly from 74.67% to 9.66%. The results in *PC* and *Dining* domains, i.e., domains with independent sub-tasks, are less affected. Nonetheless, the numbers from *Llama-3.1-70B* are even considerably lower than those from *GPT-4-turbo*, implying an enormous performance gap. This is possibly caused by the reducing parameter count of *Llama-3.1-70B* compared to *GPT-4-turbo* and *GPT-4o*.

VI. CONCLUSION

Despite their training on vast amount of data, LLMs can generate infeasible plans due to hallucinations or missing information. Moreover, classical planning techniques often require extensive annotation or domain-specific heuristics to incorporate context and semantics. 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, and validate our approach on real-world robot operations.

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] Y. Goel, N. Vaskevicius, L. Palmieri, N. Chebrolu, K. O. Arras, and C. Stachniss, "Semantically informed mpc for context-aware robot exploration," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 11 218–11 225.

[22] 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.

[23] 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.

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

[25] 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.

[26] A. Rosinol, A. Gupta, M. Abate, J. Shi, and L. Carlone, "3d dynamic scene graphs: Actionable spatial perception with places, objects, and humans," in *Robotics: Science and Systems (RSS)*, 2020.

[27] A. Rosinol *et al.*, "Kimera: From slam to spatial perception with 3d dynamic scene graphs," *The International Journal of Robotics Research*, vol. 40, no. 12-14, pp. 1510–1546, 2021.

[28] N. Hughes, Y. Chang, and L. Carlone, "Hydra: A real-time spatial perception system for 3D scene graph construction and optimization," 2022.

[29] J. Wald, H. Dhamo, N. Navab, and F. Tombari, "Learning 3d semantic scene graphs from 3d indoor reconstructions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[30] S.-C. Wu, J. Wald, K. Tateno, N. Navab, and F. Tombari, "Scenegraphfusion: Incremental 3d scene graph prediction from rgb-d sequences," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 7515–7525.

[31] Z. Wang, B. Cheng, L. Zhao, D. Xu, Y. Tang, and L. Sheng, "Vl-sat: Visual-linguistic semantics assisted training for 3d semantic scene graph prediction in point cloud," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2023, pp. 21 560–21 569.

[32] S. Koch, P. Hermosilla, N. Vaskevicius, M. Colosi, and T. Ropinski, "Sgrec3d: Self-supervised 3d scene graph learning via object-level scene reconstruction," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2024.

[33] S. Koch, P. Hermosilla, N. Vaskevicius, M. Colosi, and T. Ropinski, "Lang3dsg: Language-based contrastive pre-training for 3d scene graph prediction," *arXiv preprint arXiv:2310.16494*, 2023.

[34] Q. Gu *et al.*, "Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning," *arXiv preprint arXiv:2309.16650*, 2023.

[35] H. Chang *et al.*, "Context-aware entity grounding with open-vocabulary 3d scene graphs," in *7th Annual Conference on Robot Learning*, 2023.

[36] S. Koch, N. Vaskevicius, M. Colosi, P. Hermosilla, and T. Ropinski, "Open3dsg: Open-vocabulary 3d scene graphs from point clouds with queryable objects and open-set relationships," *arXiv preprint arXiv:2402.12259*, 2024.

[37] Z. Seymour, N. C. Mithun, H.-P. Chiu, S. Samarasekera, and R. Kumar, "Graphmapper: Efficient visual navigation by scene graph generation," in *2022 26th International Conference on Pattern Recognition (ICPR)*. IEEE, 2022, pp. 4146–4153.

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

[39] Y. Yang, S. Xiong, A. Payani, E. Shareghi, and F. Fekri, "Harnessing the power of large language models for natural language to first-order logic translation," *arXiv preprint arXiv:2305.15541*, 2023.

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

[41] 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.

[42] 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.

[43] 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.

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

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

[46] 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.