2025 IEEE International Conference on Robotics and Automation (ICRA)
May 19-23, 2025. Atlanta, USA

Large Language Model based Autonomous Task Planning for Abstract Commands

Seokjoon Kwon[1], Jae-Hyeon Park[2], Hee-Deok Jang[1], CheolLae Roh[2], and Dong Eui Chang[1*]

2025 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3315-4139-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICRA55743.2025.11128343

***Abstract*— Recent advances in large language models (LLMs) have demonstrated exceptional reasoning capabilities in natural language processing, sparking interest in applying LLMs to task planning problems in robotics. Most studies focused on task planning for clear natural language commands that specify target objects and their locations. However, for more user-friendly task execution, it is crucial for robots to autonomously plan and carry out tasks based on abstract natural language commands that may not explicitly mention target objects or locations, such as 'Put the food ingredients in the same place.' In this study, we propose an LLM-based autonomous task planning framework that generates task plans for abstract natural language commands. This framework consists of two phases: an environment recognition phase and a task planning phase. In the environment recognition phase, a large vision-language model generates a hierarchical scene graph that captures the relationships between objects and spaces in the environment surrounding a robot agent. During the task planning phase, an LLM uses the scene graph and the abstract user command to formulate a plan for the given task. We validate the effectiveness of the proposed framework in the AI2THOR simulation environment, demonstrating its superior performance in task execution when handling abstract commands.**

I. INTRODUCTION

As large language models (LLMs) have demonstrated advanced reasoning capabilities in language processing [1], [2], research on task planning in robotics using LLMs has gained considerable attention [3]. For a robot to autonomously plan and execute tasks based on user commands without intervention, both effective environment representation and task planning technologies are essential. Consequently, there has been active research into techniques for generating structured environment representations, as well as task planning methods. A common representation used in task planning with LLMs is the scene graph [4].

Early work on scene graphs focused on representing relationships between objects detected in a single image, where objects are represented as nodes and spatial relationships between them as edges [4], [5], [6], [7], [8]. As the potential of scene graphs became evident, research into 3D scene graphs for representing 3D environments gained momentum [9], [10], [11], [12]. With the development of large vision

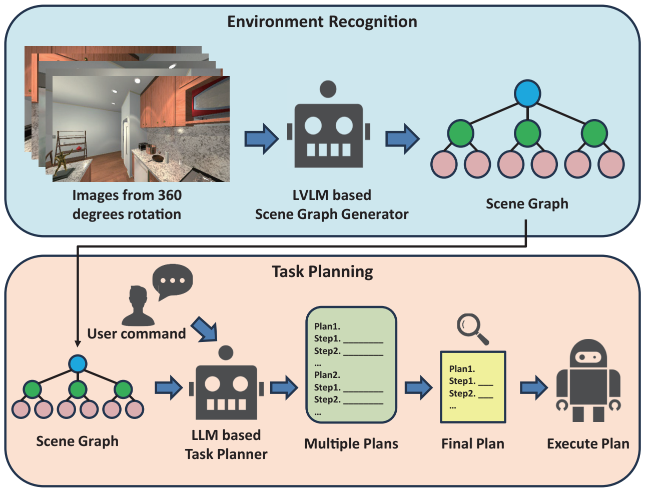


Fig. 1: Overall structure of the proposed framework. The framework consists of an environment recognition phase that generates a scene graph, and a task planning phase that receives the scene graph and an user command to generate a task plan.

language models (LVLMs), a 3D scene graph with rich semantic information by collecting captions describing each detected object was developed [12]. Although these advancements contributed to effective environment representation, the scene graphs typically included only spatial relationships between objects and did not capture inclusion relationships between objects and smaller spaces in a room, as all objects were treated within the same hierarchy. For instance, in the case of an apple on a desk, both the desk and the apple were treated as separate objects, even though the desk serves as a container for other objects, including the apple. While some studies developed frameworks to construct 3D scene graphs with hierarchical layers, including objects, rooms, and buildings [9], [10], [11], the objects within each room still existed within the same hierarchy, and the scene graphs only represented spatial relationships. They did not account for inclusion relationships between objects and spaces, such as desks or tables that could contain smaller objects. This limitation makes it difficult to plan tasks for abstract commands like 'Put the food ingredients in the same space' based on these scene graphs. As a result, studies on task planning using scene graphs [12], [13] often focused on using these scene graphs on tasks involving specific actions related to predefined objects, such as locating a particular object in the environment. However, many human commands are abstract, and task planning for such commands requires reasoning

* Corresponding author

This study was supported by Samsung Display Co., Ltd. (Project No. IAP2404168).

[1]Seokjoon Kwon, Hee-Deok Jang, and Dong Eui Chang are with the School of Electrical Engineering, KAIST, Daejeon, 34141, Republic of Korea. {jun115533, jhd6844, dechang}@kaist.ac.kr
[2]Jae-Hyeon Park and CheolLae Roh are with Mechatronics Technology Research Center, Samsung Display Co., Yongin-si 17113, Republic of Korea. {jaehyeon.park53,cl.roh}@samsung.com

based on inclusion relationships between objects and spaces. While some research has incorporated environment representations that include these relationships for task planning, these studies assumed that the environment representation is pre-given [14], [15].

Another cutting-edge element for task planning is the use of LLMs. Huang et al. [16] showed that language models can function as zero-shot planners. Singh et al. [17] explored the potential of language models for task planning in the form of Pythonic code. As the field continued to develop, there has been an expansion into studies such as [18], [19], which used feedback loops on LLM outputs to enhance the stability of task planning, as well as studies that incorporated traditional task planning methods [20], [21], such as the planning domain definition language [22]. However, most of these studies focused on tasks where the names of the target objects and the specific locations to which the objects must be moved are clearly specified [23], [24], [25], [26], [17], [27]. For instance, Ding et al. [23] addressed tasks like 'Set the table with a fork and knife,' while Song et al. [24] and Min et al. [25] dealt with tasks such as 'Place the warm cup in the cabinet' and 'Drop the pan on the table,' respectively. For robots to perform tasks in a more human-friendly manner, they must also be capable of handling abstract commands where the target objects or locations are not explicitly specified, such as 'Put the food ingredients in the same place.'

In this study, we propose an LLM-based autonomous task planning framework that includes the generation of a scene graph and a task planner for abstract commands. As illustrated in Fig. 1, the proposed framework consists of an environment recognition phase, which structures the robot's surrounding environment into a hierarchical scene graph of room-workspace-object, and a task planning phase, which generates a task plan for a user command. In the environment recognition phase, a large vision language model is employed to autonomously generate a scene graph by identifying objects and the workspaces to which these objects belong in images of the environment. Workspaces are defined as surfaces or spaces smaller than a room that can contain objects. For example, if an apple is placed on a desk, the apple is defined as an object, and the desk is defined as the workspace that contains the apple. This scene graph represents inclusion relationships between objects and spaces within the environment by introducing an intermediate layer of workspaces between the room and the objects. The scene graph is then passed to the task planning phase, where an LLM generates multiple plans for executing an abstract user command. The plan most aligned with the user's intent is selected from the generated plans and converted into executable code. Through this process, the framework can perform tasks based on abstract user commands that do not explicitly include object names and locations. We validate the proposed framework in the AI2THOR simulation environment [28], demonstrating its superior task performance and efficiency when handling abstract commands.

II. METHOD

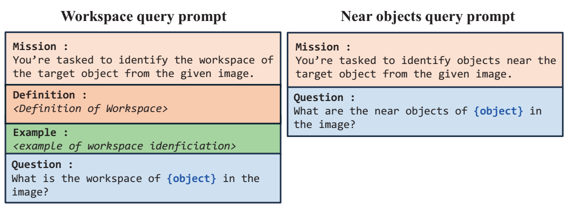
A. Simulator

We utilize the AI2THOR simulator [28], which consists of various rooms and objects, as well as one or more robot agents capable of interacting with objects. This simulator has been frequently used in research on task planning with LLMs [21], [24], [29], [30]. In this study, we use AI2THOR's iTHOR environment, which comprises a total of 120 rooms, including 30 each for kitchens, bedrooms, bathrooms, and living rooms. iTHOR provides built-in functions that allow control of the robot agent's actions, which we combine to implement three functions: `Pickup_Object`, `Put_Object`, and `GoTo`. The `Pickup_Object` and `Put_Object` functions take the name of an object as a parameter, allowing the robot agent to pick up or place the object accordingly. The `GoTo` function takes the name of an object or workspace as a parameter, generates a path from the robot agent's initial position to the target location using the coordinates of the object or workspace, and then guides the robot along the path by generating a sequence of iTHOR commands, such as `RotateRight`, `RotateLeft`, and `MoveAhead`. The A* algorithm [31] is employed for path generation. In iTHOR, the following information can be obtained: the world coordinates corresponding to specific pixels in the image captured by the robot agent, as well as the locations in the environment where the robot can move without collision. Additionally, the names and locations of workspaces and objects in the room, as well as the inclusion relationships between each workspace and object, can be accessed for evaluation purposes.

B. Environment Recognition

The goal of the environment recognition phase is to autonomously generate an object information dictionary and a scene graph. The object information dictionary is defined as data consisting of the names and location coordinates of individual objects. After generating the object information dictionary, a scene graph is created. The scene graph follows a hierarchical structure, with the room as the top-level node, workspaces as the child nodes of the room, and objects as the child nodes of the workspaces.

1) Object Information Dictionary: The process of generating the object information dictionary is as follows. At the start of the framework, the robot agent with a camera rotates 360 degrees from its initial position, rotating by r degrees at each step, and performs object detection on the RGB image at each rotation. The object detector predicts the bounding boxes and names of the objects in the image and collects point cloud data corresponding to the pixels inside the bounding boxes. The object detector used is YOLO [32] version 8, trained on a custom iTHOR image dataset. For objects detected across multiple images with the same name, the method proposed in [12] is applied to distinguish between objects by measuring geometric and semantic similarities to assess object identity. If objects with the same name are determined to be the same physical object, their point clouds


Workspace query prompt

```
Mission :
You're tasked to identify the workspace of
the target object from the given image.
Definition :
<Definition of Workspace>
Example :
<example of workspace idenficiation>
Question :
What is the workspace of {object} in the
image?
```

Near objects query prompt

```
Mission :
You're tasked to identify objects near the
target object from the given image.
Question :
What are the near objects of {object} in
the image?
```

Fig. 2: Prompts with questions for obtaining workspaces and near objects information.

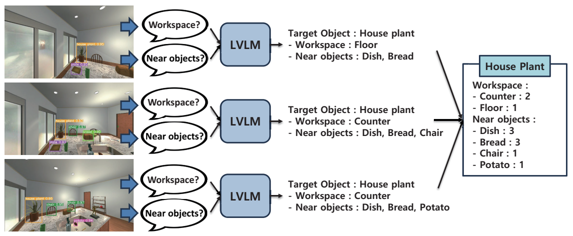


Fig. 3: Process for deriving a spatial relation dictionary.

are merged. If objects with the same name are different, they are differentiated by appending suffixes such as _1, _2, and so on. The average coordinate of each object's point cloud is considered its location. Through this process, the object information dictionary is generated, containing the names and locations of individual objects present in the environment.

2) Scene Graph: In the first stage of scene graph generation, by iterating over the images in which objects are detected by the object detector, two sequential questions are asked to the LVLM for each object present in each image. These questions aim to obtain information about the workspace and near objects for each detected object in the images. The prompt given to the LVLM for each question is illustrated in Fig. 2, where the name of the detected object is inserted into the object name section marked in blue. The LVLM processes the images and outputs the names of workspaces and near objects associated with the detected object. The number of mentions of each workspace and near object is aggregated across images for the detected object. We refer to this information as the spatial relation dictionary of the detected object. An example of deriving a spatial relation dictionary is shown in Fig. 3, with a house plant that appears in three images. For the workspace, the counter is mentioned twice, and the floor is mentioned once. For near objects, the dish and bread are each mentioned three times, while the chair and potato are each mentioned once. This information is summarized into a spatial relation dictionary, as shown in the rightmost box of Fig. 3. After generating the spatial relation dictionaries for all detected objects, we divide the detected objects into two sets based on whether a dominant workspace exists. The first set consists of detected objects with the dictionaries in which one workspace has a higher mention count than the others. For each detected object in this set, we remove the other workspaces and retain only the one with the highest mention count. The second set consists of detected objects with the dictionaries where multiple workspaces have the same mention count.

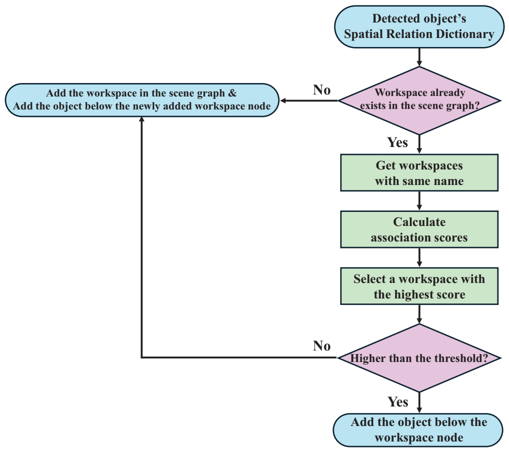
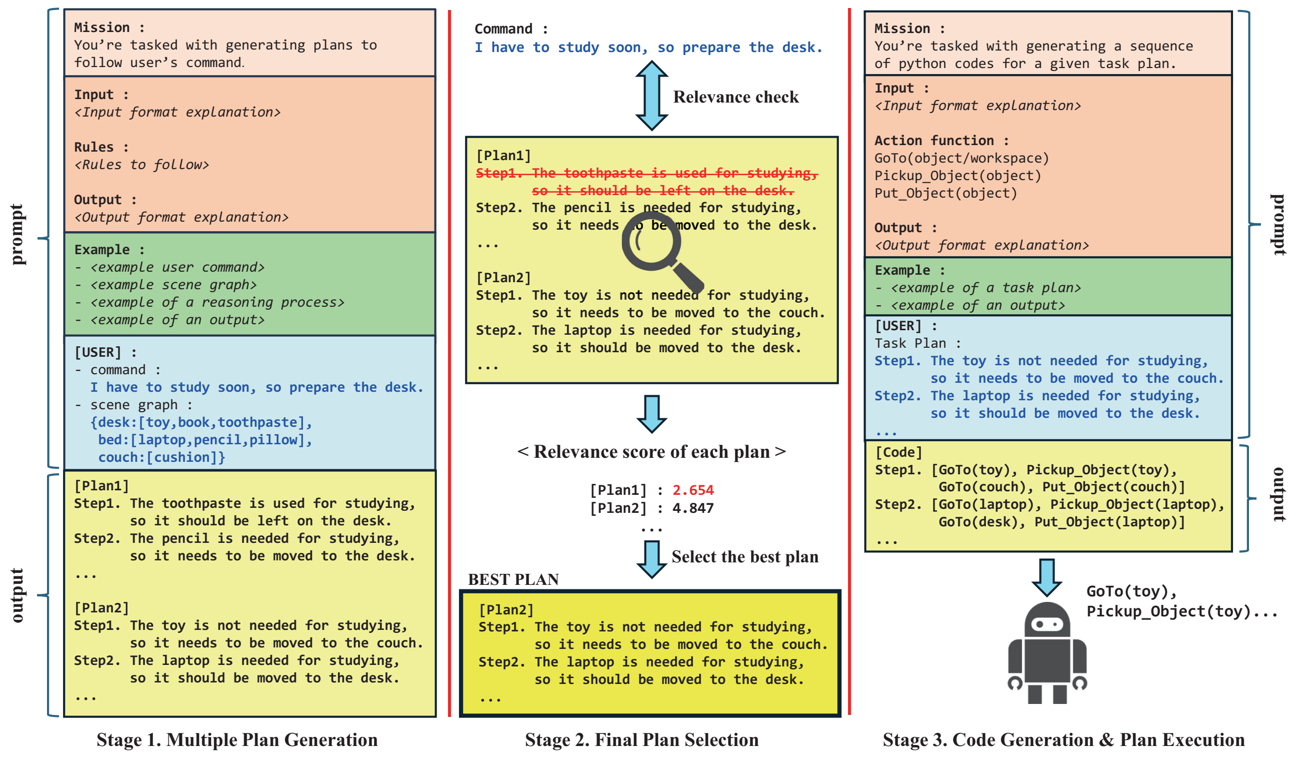

Detected object's Spatial Relation Dictionary
Workspace already exists in the scene graph?
No
Add the workspace in the scene graph & Add the object below the newly added workspace node
Yes
Get workspaces with same name
Calculate association scores
Select a workspace with the highest score
Higher than the threshold?
No
Yes
Add the object below the workspace node

Fig. 4: Generation of an initial scene graph.

We use the first set to generate an initial scene graph, as outlined in Fig. 4. We begin with a scene graph that contains only room nodes and refer to the objects of the first set one by one to execute the following processes. If the workspace name of the detected object does not exist in the scene graph, the workspace is added below the room node, and the detected object is added below this workspace node. If workspaces with the same name already exist in the scene graph, we must handle cases where the new workspace is actually different from the existing workspace nodes. In such cases, we calculate the association score between the detected object and each existing workspace node with the same name. The association score is defined as the sum of the mention count of the workspace in the detected object's spatial relation dictionary and the mention counts of near objects of the detected object that are below the existing workspace node with the same name. After calculating the association scores, the highest score is compared to a predefined threshold. If the highest score exceeds the threshold, the detected object is added below the existing workspace node with the highest score. If the scores do not exceed the threshold, a new workspace with the same name is created in the scene graph, and the object is added below that workspace node. In this case, suffixes such as _1, _2, etc., are appended to the workspace names to distinguish the new workspace from existing workspaces with the same name in the scene graph.

After generating an initial scene graph with the first set, we add the detected objects from the second set to the scene graph one by one using the following procedures. If none of the workspaces for the detected object exist in the scene graph, one of the workspaces is selected randomly and added below the room node, and the detected object is added below this workspace node. If some workspace names already exist in the scene graph, a process similar to that used for the first set is executed for those workspaces. Once the scene graph generation is complete, we add the workspace names and locations to the object information dictionary created in Section II-B.1. A workspace's location is calculated as the average of the point clouds of the objects below this workspace node. The names and locations of objects and


prompt
output
Relevance check
< Relevance score of each plan >
Select the best plan
BEST PLAN
prompt
output
Stage 1. Multiple Plan Generation
Stage 2. Final Plan Selection
Stage 3. Code Generation & Plan Execution

```
Mission :
You're tasked with generating plans to
follow user's command.

Input :
<Input format explanation>

Rules :
<Rules to follow>

Output :
<Output format explanation>

Example :
- <example user command>
- <example scene graph>
- <example of a reasoning process>
- <example of an output>

[USER] :
- command :
  I have to study soon, so prepare the desk.
- scene graph :
  {desk:[toy,book,toothpaste],
   bed:[laptop,pencil,pillow],
   couch:[cushion]}

[Plan1]
Step1. The toothpaste is used for studying,
       so it should be left on the desk.
Step2. The pencil is needed for studying,
       so it needs to be moved to the desk.
...

[Plan2]
Step1. The toy is not needed for studying,
       so it needs to be moved to the couch.
Step2. The laptop is needed for studying,
       so it should be moved to the desk.
...
```

```
Command :
I have to study soon, so prepare the desk.
```

[Plan1]
~~Step1. The toothpaste is used for studying, so it should be left on the desk.~~

```
Step2. The pencil is needed for studying,
       so it needs to be moved to the desk.
...

[Plan2]
Step1. The toy is not needed for studying,
       so it needs to be moved to the couch.
Step2. The laptop is needed for studying,
       so it should be moved to the desk.
...
```

```
[Plan1] : 2.654
[Plan2] : 4.847
...
```

```
[Plan2]
Step1. The toy is not needed for studying,
       so it needs to be moved to the couch.
Step2. The laptop is needed for studying,
       so it should be moved to the desk.
...
```

```
Mission :
You're tasked with generating a sequence
of python codes for a given task plan.

Input :
<Input format explanation>

Action function :
GoTo(object/workspace)
Pickup_Object(object)
Put_Object(object)

Output :
<Output format explanation>

Example :
- <example of a task plan>
- <example of an output>

[USER] :
Task Plan :
Step1. The toy is not needed for studying,
       so it needs to be moved to the couch.
Step2. The laptop is needed for studying,
       so it should be moved to the desk.
...

[Code]
Step1. [GoTo(toy), Pickup_Object(toy),
        GoTo(couch), Put_Object(couch)]
Step2. [GoTo(laptop), Pickup_Object(laptop),
        GoTo(desk), Put_Object(laptop)]
...
```

```
GoTo(toy),
Pickup_Object(toy)...
```

Fig. 5: Illustration of a task planning phase. It consists of three stages which are multiple plan generation stage, final plan selection stage, code generation and plan execution stage.

workspaces in the object information dictionary are used as input for the GoTo function during task execution.

C. Task Planning

In the task planning phase, given the user command, we use three stages to output the best task plan in robot code. These stages are multiple plan generation, final plan selection, and plan execution after code generation as illustrated in Fig. 5. Detailed prompts given to the LLM are uploaded on our GitHub page: https://github.com/joon725/LLM-Planner-for-Abstract-Commands/. In the first stage, which is called multiple plan generation, we transform the scene graph from Section II-B.2 into a Python dictionary form: $\{\text{key}_1\text{:value}_1, \text{key}_2\text{:value}_2, \cdots, \text{key}_N\text{:value}_N\}$, where N is the total number of workspace nodes in the scene graph, key_i is the i^{th} workspace node, and value_i is the list of object nodes below the i^{th} workspace node. An example of this form is shown in the middle left blue box of Fig. 5. Next, we create a prompt, as shown on the left side of Fig. 5, using the user command and the transformed scene graph in the [USER] section of the prompt. The LLM receives this prompt as input and generates two or more multiple plans, with each plan consisting of several steps to execute the command individually. The maximum number of plans is not set and varies each time the LLM generates multiple plans. In Fig. 5, we display two plans for instructional clarity.

In the second stage, which is called final plan selection, we quantitatively calculate the degree of relevance between each step of the plans and the user command. Let us define P as the total number of plans generated by the LLM. Let $S_p, (p \in 1, 2, \ldots, P)$ represent the total number of sequential steps for the p^{th} plan, and $L_{p,s} \in \mathcal{L}, (s \in 1, 2, \ldots, S_p)$ represent the natural language sentence of the s^{th} step in the p^{th} plan, where $\mathcal{L}$ is the set of all natural language sentences. $L_u \in \mathcal{L}$ is defined as the user command, and $f(\cdot) : \mathcal{L} \rightarrow \mathbb{R}^{768}$ as the text feature extractor of the vision-language model BLIP [33]. We define the relevance score $\text{RS}_{p,s}$ of $L_{p,s}$ and L_u as follows:

$$\text{RS}_{p,s} = \frac{f(L_{p,s}) \cdot f(L_u)}{\|f(L_{p,s})\| \|f(L_u)\|},$$

which represents the cosine similarity between the text feature vector of $L_{p,s}$ and L_u, where $\| \cdot \|$ denotes the Euclidean norm of a vector. After calculating all the relevance scores, we remove the steps with a relevance score below a predefined threshold from the plans. This threshold is set to 0.5, where the number of 0.5 has been selected as an optimal one after several empirical trials. Through this process, steps in the plans with a low degree of semantic relevance to user's command are eliminated. Next, we calculate a relevance score for each plan by summing the relevance scores of the remaining steps in that plan. The plan with the highest relevance score is selected, as it is the most relevant to the user's command among the generated plans.

In the final stage, which is called plan execution after code generation, we create a prompt as shown in the right part of Fig. 5, with the selected plan in the [USER] section. The LLM receives this prompt as input and generates code using the action functions from Section II-A. By executing this

TABLE I: Rooms, workspaces, and object categories of the iTHOR environment.

Room	Workspace	Object category
Kitchen	Dining table Counter	Food ingredients Tableware Cooking utensils
Bedroom	Desk Bed	Electronics Sports equipment

code step by step from top to bottom, the iTHOR robot agent carries out the command.

III. EXPERIMENT

A. Experimental Setup and Results

In this study, experiments are conducted in the iTHOR environment across a total of 60 rooms, with 30 kitchen rooms and 30 bedroom rooms. The rooms, workspaces, and objects in the iTHOR environment are tabulated in Table I. Since there are a multitude of objects, we classify them into five categories based on their usage: food ingredients, tableware, cooking utensils, electronics, and sports equipment. The LVLM used in the environment recognition phase utilizes OpenAI's GPT-4o [34] API, while the LLM used in the task planning phase employs OpenAI's GPT-4 [35] API. The association score threshold for the scene graph is set to 4 for the kitchens and 1 for the bedrooms, after conducting experiments with thresholds ranging from 1 to 20 and selecting the values that yielded the highest average v-measure [36], a well-known clustering metric, for each environment.

We design four types of tasks with abstract commands of increasing difficulty, as shown in Table II. Task 1 is composed of tasks such as 'Place the food ingredients on the dining table,' where objects from a single object category need to be moved to a specific workspace. Task 2 is composed of tasks like 'Gather the food ingredients in the same location,' where the goal is to gather objects from a single object category into the same workspace, which is not explicitly specified. Task 3 is composed of tasks like 'Place the food ingredients on the dining table and the tableware on the counter,' which requires placing objects from two object categories at two different workspaces respectively. Task 4 is composed of tasks like 'Place the food ingredients on the dining table and move the objects that are not food ingredients away from the dining table,' where objects from a single object category must be placed in a specific workspace, while all other objects not belonging to that category must be moved to different workspaces. Task 1, Task 2, Task 3, and Task 4 are run 60, 32, 25, and 46 times, respectively, using various combinations of rooms, workspaces, and object categories. On average, each task contains 2.02, 2.3, 3.58, and 4.52 objects from the target object category in the environment. The actual total number of objects in each task is higher than these numbers, as there are also objects from other categories.

We devise the two metrics of executability and average accuracy to best describe the performance of autonomous task planning, which are adapted from the metrics used in

TABLE II: Four types of tasks.

Task	Command
Task 1	Place an {object category} at a specific {workspace}
Task 2	Gather an {object category} at the same place
Task 3	Place two {object categories} at two different {workspaces} respectively
Task 4	Place an {object category} at a specific {workspace}, and move the objects that are not in the category away from that {workspace}

[17], [18], [20], [21], [24]. Executability (%) is defined as the number of successfully executed tasks divided by the total number of task runs. When the LLM generates steps involving things not present in the environment, the plan including those steps cannot be executed. High executability can be interpreted as having a prompt with appropriate instructions and input information, or as having a good process for eliminating infeasible steps. Accuracy is calculated for each task run and is defined as the ratio of correctly moved objects in the chosen object category. If an object is not moved or is moved to the wrong workspace, it is considered a failure. By averaging the accuracies, we obtain the average accuracy.

For comparison, two other task planning methods using LLM are considered. The first method is SayPlan [14], which utilizes a scene graph and LLM. It generates a single plan based on a reduced scene graph, created through a semantic search step that selects only the nodes relevant to the task from the scene graph. Both the semantic search step and task planning are performed by the LLM, and we implement the prompts as proposed in [14], except for any formal mismatches. To ensure a fair comparison, the feedback loop for revising plans is excluded, and the plan is generated only once. The second method is the Naïve LLM Planner, proposed in [16], which generates a single plan using the LLM. In this method, the LLM generates each step of the plan sequentially, taking the previous steps as input to generate the next step. For both methods, OpenAI's GPT-4 API is used.

The experimental results for our framework, SayPlan, and the Naïve LLM Planner are presented in Table III. For the four tasks, the average value for each metric is compared. The proposed framework shows the best performance in both of executability and average accuracy. For executability, the proposed framework outperforms SayPlan by 10.27%p and the Naïve LLM Planner by 27.42%p, where the symbol of %p denotes the percentage point. For average accuracy, the proposed framework outperforms SayPlan by 5.25%p and the Naïve LLM Planner by 29.23%p. These experimental results indicate that the proposed framework accomplishes the tasks most successfully for abstract commands among the compared methods.

B. Ablation Study

To analyze the influence of the proposed scene graph structure, an ablation study is conducted. The variant (Ours with general SG) uses a general scene graph that contains only spatial relationships between objects without an intermediate layer of workspaces. For this scene graph, we refer to the method proposed in [12], which generates a scene graph containing nodes for objects and edges for representing the

TABLE III: Results for task planning.

	Task	Ours	SayPlan	Naïve LLM Planner	Ours with general SG
Executability (%)	Task 1	90	87.5	55	88.3
	Task 2	87.5	84.38	87.5	80
	Task 3	76	60	40	72
	Task 4	95.23	75.76	56.52	73.81
	Avg	87.18	76.91	59.76	78.53
Average accuracy (%)	Task 1	75.6	74.58	25.83	72.22
	Task 2	74.22	72.4	61.2	64.58
	Task 3	54.93	53.07	33.87	51.6
	Task 4	68.56	52.27	35.48	46.16
	Avg	68.33	63.08	39.1	58.64

spatial relationships between the objects. The results of the ablation study are shown in the rightmost column of Table III. For the both metrics, the originally proposed framework demonstrates the best performance. For executability, the proposed framework outperforms the variant by 8.65%p. In terms of average accuracy, our framework achieves 9.69%p higher than the variant. The comparison of the original framework with the variant shows that implementing our scene graph structure improves the accuracy of task planning.

C. Experiments with Highly Abstract Commands

To showcase the framework's ability to perform tasks based on more abstract commands, additional experiments are conducted. These experiments take place in one of the kitchens and one of the bedrooms. For each room, the user commands 'Prepare for cooking before I start cooking for dinner.' and 'I am going to sleep soon, prepare for it.' are given, respectively. These commands convey only the user's intent without specifying any target objects or locations, making them highly abstract. For the first command, the executed task plan includes moving the pot and pan from the counter to the stove burner, relocating the potato from the sink to the counter, and moving the bread from the kitchen island to the counter. The task execution process for this command is illustrated in Fig. 6. For the second command, the final task plan includes moving the laptop from the bed to the desk, placing the tennis racket and basketball from the bed to the floor, and relocating the book from the bed to the desk. These plans are quite reasonable, and the results demonstrate that the proposed framework is capable of successfully executing tasks based on highly abstract commands.

IV. CONCLUSIONS

In this study, we propose an LLM-based autonomous task planning framework for abstract user commands. The framework integrates autonomous environment recognition with subsequent task planning. The scene graph generated by the LVLM during the environment recognition phase structures the environment with semantic inclusion information between objects and small spaces in the room. The task planning phase, which takes the scene graph and an abstract user command as inputs, generates multiple task plans and selects the final plan that best matches the user's command.

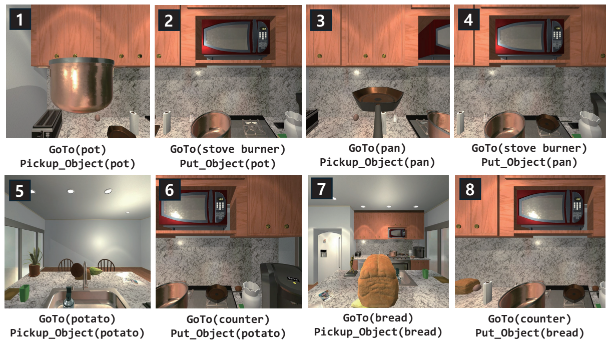


Fig. 6: Task execution for 'Prepare for cooking before I start cooking for dinner.'

The combination of the scene graph and the task planning process enables successful task execution, even for abstract commands, as validated through comparative analysis with other methods and an ablation study of variants of the proposed framework. However, the proposed framework has certain limitations. Since LVLM is used to generate the scene graph, ambiguity may arise when determining the workspace to which each object belongs. Additionally, the scene graph generation process requires multiple images from different viewpoints to accurately capture the spatial relationships, leading to potential inefficiencies in real-time applications. For future work, we aim to enhance the scene graph generation process solving the algorithmic limitations, and apply the proposed framework in real-world settings by integrating open source LLMs for task planning making the framework more accessible and practical.

REFERENCES

[1] T. B. Brown, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[2] E. Zelikman, Y. Wu, J. Mu, and N. Goodman, "Star: Bootstrapping reasoning with reasoning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 15 476–15 488, 2022.

[3] X. Huang, W. Liu, X. Chen, X. Wang, H. Wang, D. Lian, Y. Wang, R. Tang, and E. Chen, "Understanding the planning of LLM agents: A survey," *arXiv preprint arXiv:2402.02716*, 2024.

[4] J. Johnson, R. Krishna, M. Stark, L.-J. Li, D. Shamma, M. Bernstein, and L. Fei-Fei, "Image retrieval using scene graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3668–3678.

[5] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, "Scene graph generation by iterative message passing," in *Proceedings of the IEEE Conference On Computer Vision and Pattern Recognition*, 2017, pp. 5410–5419.

[6] J. Gu, H. Zhao, Z. Lin, S. Li, J. Cai, and M. Ling, "Scene graph generation with external knowledge and image reconstruction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1969–1978.

[7] Y. Li, W. Ouyang, B. Zhou, K. Wang, and X. Wang, "Scene graph generation from objects, phrases and region captions," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[8] S. Khandelwal, M. Suhail, and L. Sigal, "Segmentation-grounded scene graph generation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 15 879–15 889.

[9] I. Armeni, Z.-Y. He, J. Gwak, A. R. Zamir, M. Fischer, J. Malik, and S. Savarese, "3D scene graph: A structure for unified semantics, 3D space, and camera," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 5664–5673.

[10] A. Rosinol, A. Violette, M. Abate, N. Hughes, Y. Chang, J. Shi, A. Gupta, and L. Carlone, "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.

[11] N. Hughes, Y. Chang, and L. Carlone, "Hydra: A real-time spatial perception system for 3D scene graph construction and optimization," *arXiv preprint arXiv:2201.13360*, 2022.

[12] Q. Gu, A. Kuwajerwala, S. Morin, K. M. Jatavallabhula, B. Sen, A. Agarwal, C. Rivera, W. Paul, K. Ellis, R. Chellappa, *et al.*, "Conceptgraphs: Open-vocabulary 3D scene graphs for perception and planning," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 5021–5028.

[13] Z. Dai, A. Asgharivaskasi, T. Duong, S. Lin, M.-E. Tzes, G. Pappas, and N. Atanasov, "Optimal scene graph planning with large language model guidance," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 14 062–14 069.

[14] 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 task planning," *arXiv preprint arXiv:2307.06135*, 2023.

[15] Z. Ni, X. Deng, C. Tai, X. Zhu, Q. Xie, W. Huang, X. Wu, and L. Zeng, "Grid: Scene-graph-based instruction-driven robotic task planning," *arXiv preprint arXiv:2309.07726*, 2023.

[16] 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.

[17] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 11 523–11 530.

[18] Z. Zhou, J. Song, K. Yao, Z. Shu, and L. Ma, "ISR-LLM: Iterative self-refined large language model for long-horizon sequential task planning," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 2081–2088.

[19] S. H. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, "Chatgpt for robotics: Design principles and model abilities," *IEEE Access*, 2024.

[20] Y. Liu, L. Palmieri, S. Koch, I. Georgievski, and M. Aiello, "Delta: Decomposed efficient long-term robot task planning using large language models," *arXiv preprint arXiv:2404.03275*, 2024.

[21] A. Mavrogiannis, C. Mavrogiannis, and Y. Aloimonos, "Cook2LTL: Translating cooking recipes to LTL formulae using large language models," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 17 679–17 686.

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

[23] Y. Ding, X. Zhang, C. Paxton, and S. Zhang, "Task and motion planning with large language models for object rearrangement," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 2086–2092.

[24] 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.

[25] S. Y. Min, D. S. Chaplot, P. Ravikumar, Y. Bisk, and R. Salakhutdinov, "Film: Following instructions in language with modular methods," *arXiv preprint arXiv:2110.07342*, 2021.

[26] R. Wang, Z. Yang, Z. Zhao, X. Tong, Z. Hong, and K. Qian, "LLM-based robot task planning with exceptional handling for general purpose service robots," *arXiv preprint arXiv:2405.15646*, 2024.

[27] L. Sun, D. K. Jha, C. Hori, S. Jain, R. Corcodel, X. Zhu, M. Tomizuka, and D. Romeres, "Interactive planning using large language models for partially observable robotic tasks," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 14 054–14 061.

[28] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, M. Deitke, K. Ehsani, D. Gordon, Y. Zhu, *et al.*, "AI2-THOR: An interactive 3D environment for visual ai," *arXiv preprint arXiv:1712.05474*, 2017.

[29] S. Nayak, A. M. Orozco, M. T. Have, V. Thirumalai, J. Zhang, D. Chen, A. Kapoor, E. Robinson, K. Gopalakrishnan, J. Harrison, *et al.*, "Long-horizon planning for multi-agent robots in partially observable environments," *arXiv preprint arXiv:2407.10031*, 2024.

[30] S. S. Kannan, V. L. Venkatesh, and B.-C. Min, "SMART-LLM: Smart multi-agent robot task planning using large language models," *arXiv preprint arXiv:2309.10062*, 2023.

[31] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[32] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[33] J. Li, D. Li, C. Xiong, and S. Hoi, "BLIP: Bootstrapping language-image pre-training for unified vision-language understanding and generation," in *International Conference on Machine Learning*. PMLR, 2022, pp. 12 888–12 900.

[34] OpenAI, "Hello gpt-4o," 2024, accessed on August 30, 2024. [Online]. Available: https://openai.com/index/hello-gpt-4o/

[35] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "GPT-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[36] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," in *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, 2007, pp. 410–420.