LookPlanGraph: Embodied Instruction Following Method with VLM Graph Augmentation

Anonymous submission

***Abstract*— Methods that use Large Language Models (LLM) as planners for embodied instruction following tasks have become widespread. To successfully complete tasks, the LLM must be grounded in the environment in which the robot operates. One solution is to use a scene graph that contains all the necessary information. Modern methods rely on prebuilt scene graphs and assume that all task-relevant information is available at the start of planning. However, these approaches do not account for changes in the environment that may occur between the graph's construction and the task execution. We propose LookPlanGraph – a method that leverages a scene graph composed of static assets and object priors. During plan execution, LookPlanGraph continuously updates the graph with relevant objects, either by verifying existing priors or discovering new entities. This is achieved by processing the agent's egocentric camera view using a Vision Language Model. We conducted experiments with changed object positions VirtualHome and OmniGibson simulated environments, demonstrating that LookPlanGraph outperforms methods based on predefined static scene graphs. To demonstrate the practical applicability of our approach, we also conducted experiments in a real-world setting. Additionally, we introduce the GraSIF (Graph Scenes for Instruction Following) dataset with automated validation framework, comprising 514 tasks drawn from SayPlan Office, BEHAVIOR-1K, and VirtualHome RobotHow.**

I. Introduction

The pursuit of embodied agents, such as robots, that can understand and execute complex human instructions in dynamic environments is a fundamental goal of Embodied Artificial Intelligence. Recent advances in large language models (LM) have shown promising capabilities in natural language reasoning and planning [1], [2]. However, for effective task execution, these models must be grounded in the physical environment – often achieved through scene graphs that capture objects and their relationships [3].

Most current methods [4], [5], [6] assume a fully known environment and generate complete action sequences in a single pass. This approach significantly limits their effectiveness in dynamic environments, where objects can appear or change position over time (see Figure 1). While several methods for dynamically building graphs have been proposed [7], [8], these approaches primarily focus on vision language navigation and do not fully address manipulation challenges, such as inferring goal states from complex instructions and generating feasible plans for multiple object rearrangement.

To address these limitations, we propose LookPlanGraph, a method that integrates graph-based planning with dynamic scene updates. At the core of our approach is the **Memory Graph**, which consists of three key components: 1) the general scene structure; 2) the immovable objects (assets) previously interacted with; and 3) the probable positions of

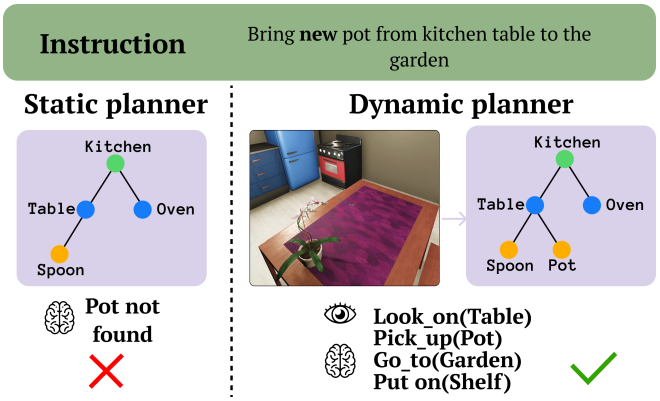

Fig. 1. Static planners rely solely on predefined scene graphs, making them ineffective when objects are missing or misplaced. In contrast, dynamic planners can explore the environment in real time and update the scene graph to account for newly discovered objects. This allows them to successfully execute tasks.

the movable objects (priors). Complementing the memory graph is the **Scene Graph Simulator** (SGS), which refines actions generated by the LM and updates the Memory Graph to reflect environmental changes, enabling the LM to dynamically choose actions based on the current state. This design enables the integration of a **graph augmentation module**, which can discover new objects or update their positions in a changed environment. The agent explores the environment using available assets, capturing images from predefined poses. These images are then processed by a Vision Language Model (VLM) to infer object relationships and properties, incorporating the detected objects into the accurate scene representation.

We comprehensively evaluate LookPlanGraph across dynamic and static scenarios, including real-world office experiments to demonstrate practical applicability. Given the lack of graph-based instruction-following datasets, we introduce GraSIF (Graph Scenes for Instruction Following), a dataset constructed with manipulation tasks from SayPlan Office [4], Behaviour-1k [9], and RobotHow [10]. Our results show that LookPlanGraph achieves competitive planning performance in static environments and outperforms existing methods in dynamic settings.

In summary, our main contributions are:

1) **LookPlanGraph:** We introduce a method for adaptive and context-aware planning that achieves the highest performance in dynamic environments.
2) **Graph augmentation module:** We propose a module that utilizes a VLM to dynamically update scene

graphs based on real-time observations.

3) **GraSIF Dataset:** We present the benchmark of 514 tasks, along with a lightweight, automated validation framework, designed to evaluate graph-based instruction following in household manipulation scenarios.

II. Related Works

A. Embodied Planning

Embodied instruction following involves generating sequences of actions to accomplish goals described in natural language within a given environment. Traditional methods follow two main approaches. *Symbolic planning*, where entities, actions, and goals are represented as discrete symbols, such as in the Planning Domain Definition Language [11] (PDDL) or Temporal Logic (TL)[12]. *Hierarchical policy learning* for grounding language instructions into structured sequences of actions [1], [13], [14]. While these methods are effective in controlled environments, they struggle with scalability and generalization. Adapting them to new tasks often requires additional analysis and training or the construction of new symbolic representations to account for changes in task instructions.

Recently, LM have gained traction in task planning [15], [16], [4], [17], [18] due to their strong in-context learning and reasoning abilities. Some approaches [5], [6], [19], combine LMs with symbolic planners by having the model generate PDDL descriptions. A major challenge in LM-based planning is grounding in the environment. Certain methods work in textual [16], [20] or simulated settings [21], [17], [22], where the environment itself provides feedback or additional context to help guide the model's output, support that may not be available in real-world scenarios. SayPlan [4] tackles this issue by constructing feedback from pre-constructed 3D scene graph. While this enhances plan executability, it assumes that the environment remains static after the map is created. This assumption limits the method's applicability in dynamic real-world settings. Our approach allows to incorporate environmental information during planning and adjusts LM chosen actions using the graph structure.

B. Dynamic Instruction Following

Several approaches [7], [8] dynamically build graphs guided by LMs. Yet, they target vision-language navigation tasks aimed mainly at locating objects and do not address tasks requiring goal state comprehension and rearrangement of multiple objects. For example, in MoMa-LM [7] manipulation is limited to open and close actions, designed to address occlusion caused by closed assets and don't involve object relocation. "Success" is defined as observing the target object and calling "done()".

VeriGraph [23] uses a VLM to construct task-specific graph representations and generate plans accordingly. While VLMs show strong potential for graph construction, VeriGraph has only been evaluated in tabletop environments, limiting its applicability to more complex scenarios. Our method extends this idea to multi-room environments, enabling a broader range of tasks using only image-based observations.

III. Problem Formulation

We tackle the problem of enabling an autonomous mobile manipulator to strategically plan, traverse, and interact with objects in domestic environments. Object locations may vary between initial environment mapping and actual task performance. The goal is to synthesize action sequences that conform to a given natural language instruction I.

To achieve this, we make two key assumptions: We assume a pre-constructed graph representation of the environment, denoted as $G = (V, E)$, where V and E represent the sets of vertices and edges, respectively. The graph G is structured hierarchically into four primary levels: scene, places, assets, and objects. The scene is represented at the top level by a single *scene* node that connects to all place nodes. Each place contains nodes for immovable assets, which in turn hold nodes for movable objects. These object nodes can be linked to their parent assets or to other objects. Both asset and object nodes have defined states and affordances. The edges E capture "inside" spatial relationship within higher-level nodes. For object-object and object-asset connections, an additional type "ontop" present. Graphs of this form can be constructed using existing techniques [24], [3], [8]. It is important to note that LookPlanGraph can function without initial information about objects, but this information is still included as it provides valuable priors for guiding the search process, potentially derived from previous task executions or the initial graph construction. We also assume that the robot has access to high-level manipulation actions $\mathbf{A}$, for which approaches from prior works [25] can be used.

Our objective is to create high-level decision-making policy denoted by $\pi(\mathbf{A} \mid I, G)$ that is capable of generating an action $\mathbf{A}$ that is executable within the environment. This policy should enable the embodied agent to follow an natural language instruction I by appropriately modifying the environment's state.

IV. Method

A. Overview

We introduce LookPlanGraph – a method that dynamically generates and refines actions using a graph representation of the scene. As illustrated in Figure 2, the process begins by copying the initial graph to a memory graph, with all object nodes initially marked as unseen. The method then proceeds with the iterative process outlined in Algorithm 1 (steps 4-14), where it repeatedly generates and executes actions in the environment.

LM Decision-Making (6): The LM is assigned the role of an agent and provided with a set of possible actions (*discover objects, go to, pick up, rearrange, open, close, turn on, turn off, done*). It receives the few-shot examples of input-output interactions, a text-serialized memory graph, instruction, previous reasons and actions, and, if available, feedback. An example prompt structure is shown in Figure 3. Based on this input, the LM generates a text description detailing the reasoning behind the decision and the subsequent action to be executed.

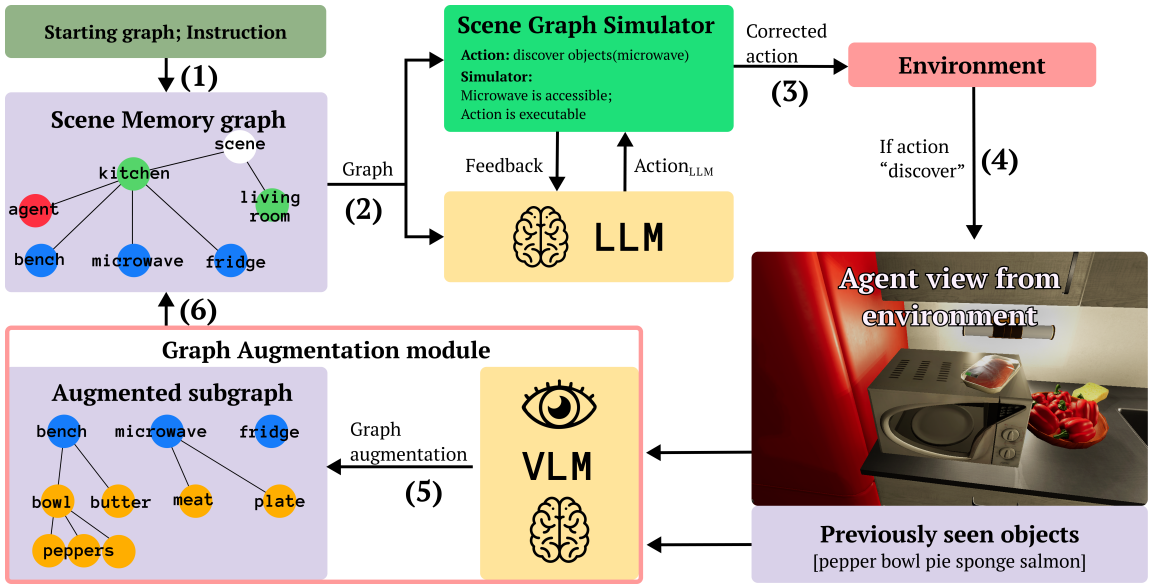

Fig. 2. **LookPlanGraph Overview:** The LookPlanGraph starts with an instruction and a static environment graph **(1)**. A scene memory graph, initially a copy of the starting graph, is processed by the LM with the task description and is also sent to the Scene Graph Simulator **(2)**. The LM suggests an action, which the Simulator checks for feasibility. If feasible, the action changes the environment and updates the SMG **(3)**. For actions requiring visual feedback (e.g., "discover_objects"), the environment sends an egocentric camera view to the VLM **(4)**. The VLM processes this image, along with previously seen objects from the SMG, to generate an augmented subgraph **(5)**, which updates the SMG **(6)**. This cycle **(2-6)** repeats until the LM decides that the task is complete.

Algorithm 1 LookPlanGraph

Require: Instruction I, Starting graph G

Modules: LM — Language Model, GAM — Graph Augmentation Module, ENV — Environment, SGS — Scene Graph Simulator

Variables: M — Scene Memory graph; F — feedback string; action$_{\text{lm}}$ — action chosen by LM; action$_{\text{corr}}$ — action corrected by SGS.

Initialize: $M \leftarrow G$

repeat

  repeat

    action$_{lm} \leftarrow$ LM(M, I, F)

    action$_{corr}, F \leftarrow$ SGS$(M,$ action$_{lm})$

  until $F == None$

  ENV(action$_{corr}$)

  if action$_{corr}$ == "discover_objects" **then**

    new_nodes $\leftarrow$ GAM(ENV, M)

    M.append(new_nodes)

  end if

until action$_{corr}$ == "done"

Simulation and Feedback (5-8): The output from the LM is sent to the SGS to verify the action's feasibility. The SGS, referencing the memory graph, either corrects the action to fit the current environment or returns it to the LM for replanning in case of hallucinations or structural errors. When the action is deemed valid, the SGS updates memory graph to reflect the changes in the environment.

Environment Interaction (9): Once validated by the simulator, the action is executed in the environment. For exploratory actions like *"discover_objects"*, the Graph Augmentation Module is triggered to expand the memory graph.

Graph Augmentation via VLM (10-13): Upon activation, the VLM processes image from agent egocentric camera, identifying objects and adding their corresponding nodes to the memory graph.

This loop continues until the LM selects the "done" action, a sequence of three infeasible actions, or upon reaching a predefined action limit. Notably, the same VLM employed for visual graph augmentation can also serve as the LM for action decision-making, enhancing operational efficiency and reducing system complexity.

B. Memory Graph

Real-world environments are inherently dynamic, meaning that object positions in a pre-constructed graph may not remain accurate during execution. To account for this, all objects in the initial graph are marked as unseen at the start of the planning process, indicating that their presence and location must be confirmed through perception.

Building a plan dynamically requires prompting the scene graph at each decision point. However, querying the entire graph for every action would incur a high interaction cost and potentially degrade the model's reasoning performance due to long prompt lengths. To mitigate this, a memory graph is used as an intermediate representation, filtering and abstracting only the most relevant information. This graph includes a list of known places, the agent node, and only those assets and unseen objects within the agent's immediate surroundings (e.g., the same room).

To maintain continuity and accumulate knowledge during task execution, the memory graph is incrementally updated

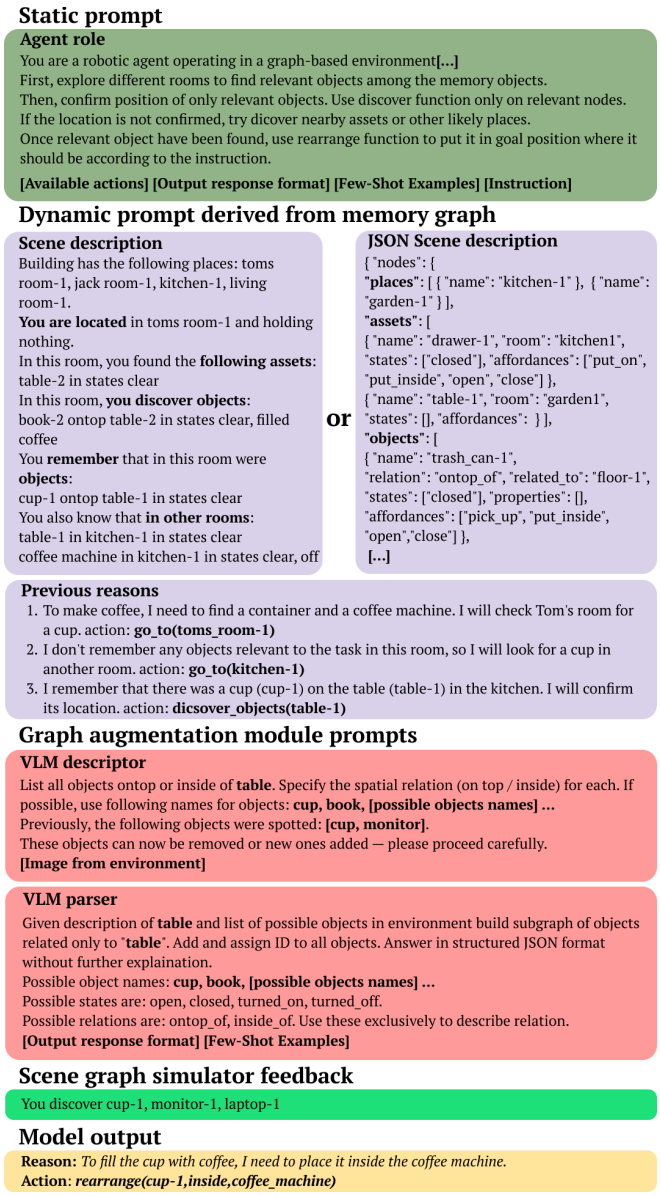

Fig. 3. LookPlanGraph prompt structure.

with all discovered objects and interacted-with assets. These elements remain visible even if the agent moves to a different location, allowing the graph to serve as a persistent memory module, supporting the agent in recalling and utilizing previously acquired information. When the graph augmentation module is used, unseen objects are removed or replaced with objects extracted from the environment. See Figure 3 for text serialized memory graph representation.

C. Scene Graph Simulator

The SGS provides a dynamic and maintained representation of the environment, ensuring that the agent remains contextually grounded during task execution. It corrects erroneous LM actions using two primary mechanisms: reprompting and rule-based correction. Errors typically fall into two categories: 1) hallucinations, which occur when the LM references non-existent objects or actions, and response structure errors; and 2) execution errors, which occur when the agent attempts infeasible actions in the current environment, such as picking up an item from a closed container.

To address hallucinations, the SGS uses a reprompting strategy similar to SayPlan [4], providing the LM with corrective feedback. For execution errors, it applies rule-based adjustments to reduce costly reprompting. For example, if the LM suggests *pick up(apple-1)* while the apple is in a closed container, the SGS can automatically add *open(container)* as a prerequisite. We extend this approach by introducing a higher-level "rearrange" action for the LM, allowing the SGS to autonomously generate intermediate steps without requiring the explicit generation of each individual action.

D. Graph Augmentation Module

For visual grounding within the environment, the LM can use the "discover_objects" action to inspect assets or objects, adding identified objects to the memory graph. An example of this interaction is shown in the lower part of Figure 2. The procedure starts by capturing an image from the agent's current field of view. This image, along with a list of known object names in the scene, and prior information about objects is passed to VLM. The VLM is instructed to describe the scene and detect objects, their states, and their spatial relationships to the observed asset. This description then parsed into a structured format. The identified nodes are incorporated into the memory, making them accessible for subsequent interactions. The prompts for the VLM is provided in Figure 3.

V. The GraSIF Dataset

Planners that utilize graph scene representations are gaining popularity in the research community. However, the lack of publicly available task-specific scene graph datasets, or reliance on computationally intensive physics simulators, complicates the evaluation of planning methods. As a result, many researchers resort to proprietary data, hindering consistent comparison across approaches. To address this gap, we introduce GraSIF (Graph Scenes for Instruction Following) – a benchmark comprising 514 instruction-following tasks represented as scene graphs in household environments, paired with a lightweight automated validation framework. GraSIF integrates textual instructions and environmental data from three sources: BEHAVIOR-1K [9], VirtualHome RobotHow [10], and SayPlan Office [4]. It focuses specifically on *mobile manipulation* tasks, where an agent must find a sequence of manipulation actions to complete a given instruction. For each environment, GraSIF provides unambiguous natural language instructions along with both the initial and goal scene graphs, representing the state before and after task execution. All graphs are constructed using consistent filtering rules to ensure comparability across tasks.

In VirtualHome RobotHow, we retained only manipulation-relevant tasks while removing tasks like *Take a shower* that not relevant for robot platform. Items were categorized via the *grabbable* property into objects

TABLE I
THE GRASIF DATASET STATISTICS.

Subdata	Tasks	Rooms	Nodes	Actions
SayPlan Office	29	**37**	**202.6**	4.2
BEHAVIOR-1K	177	1.23	12.1	**9.8**
VirtualHome	**308**	4	195.7	3.1

and assets. Task names were disambiguated, e.g., "Turn on light" → "Turn on top light in kitchen." In BEHAVIOR-1K, graphs were derived from PDDL descriptions with annotated object states and relations. To keep graphs compact, we focused on tasks that require manipulation predicates such as ontop and inside, excluding irrelevant cooking or cleaning tasks. We also create a natural language description for each task based on its goal graph representation. For SayPlan Office, we reconstructed graphs from the original paper using task descriptions and figures. Ambiguous instructions were paraphrased and goal graphs were generated with a scene graph simulator based on the action sequences provided in the paper.

To evaluate action sequences generated by planners, we first identify task-relevant nodes whose position or state changes between the initial and goal states. We then run the planner's sequence in a scene graph simulator and compare the resulting goal state to the expected state of these changed nodes. Since some tasks can have unambiguous but multiple valid solutions (e.g., bringing any three bottles to a table), we validate by checking the required number of relations for key objects rather than exact IDs. The precision of the plan is calculated by comparing the number of correctly placed object nodes and assets states to the expected in the goal state. A plan is considered successful if all objects are in the correct positions, assets are in the desired states, and the plan does not contain any infeasible actions. This approach enables fast automated validation without relying on complex physics simulations.

In total, the GraSIF dataset comprises 10 diverse scenes and 514 tasks, each emphasizing different aspects of embodied planning. The largest environment, SayPlan Office, spans 37 rooms, providing a complex and extensive planning space. In contrast, BEHAVIOR-1K tasks, while typically set in smaller, one-to-two-room environments, feature long-horizon tasks that can require up to 40 actions to complete. The VirtualHome environment consists of 4 rooms and includes 195 object nodes, resulting in a densely populated setting.

The characteristics of the integrated datasets are summarized in Table I, where the values for "Rooms", "Nodes", and "Actions" represent the mean across all tasks.

VI. EXPERIMENTS

A. Baselines

We evaluate LookPlanGraph against five baseline methods that incorporate LM as graph planners. **LLM-as-P** [5] produces a complete sequence of actions by utilizing the task description, the available actions within the environment, and a one-shot example. **LLM+P** [5] translates natural language task descriptions into PDDL problem formulations and then relies on a classical planner to generate plans. In practice, this approach struggles with syntactic and semantic errors in LLM-generated PDDL, leading to parsing failures. To overcome this, instead of using the original Manipulation domain, we designed a domain specifically tailored to our graph-based environment, enabling evaluation of instruction-following tasks. SayPlan [4] operates in two stages: first, it extracts a task-relevant subgraph using LM-based semantic search, and then it generates a full plan over this subgraph, iteratively refining it based on scene graph feedback. Due to the lack of open-source code, we created our own implementation for this study. However, observing frequent stage confusion in smaller language models, we developed **SayPlan Lite**. This variant enforces a strict separation between its search and planning phases: only the search API is available during the search phase, and only the planning API during the planning phase. Furthermore, the action set is simplified by replacing the original *access* and *release* operations with the more direct *put on* and *put inside* actions, which helps to unify the low-level action set across all methods. **ReAct** [16] is a dynamic approach that uses feedback from the simulation as exploration and saves full interactions history during planning. To incorporate ReAct into graph planning, we adapt the SayPlan simulator to provide feedback on each action in the form of an ALFWorld [26] textual environment, as it was implemented in the original paper. For tasks requiring perception, we employ our graph augmentation module.

B. Experimental setup

To assess LookPlanGraph's **performance in dynamic environments** we conducted experiments using VirtualHome and OmniGibson simulators. In VirtualHome, we designed 50 tasks across three categories: placing an object in a refrigerator, heating an object in a microwave, and washing dishes in a dishwasher. OmniGibson does not provide native support for high-level actions; therefore, a scene graph simulator was utilized to emulate these actions within the environment. For perception actions, we collected images from simulations for each asset containing the required objects, using 50 tasks from the Behaviour-1K dataset. To introduce dynamic changes, we modified the positions of objects in the initial scene graph for half of the tasks in both environments.

Graph augmentation capabilities were evaluated using the images gathered during simulation. We measured accuracy with the F1-score by comparing the nodes and edges of the generated scene graphs against ground truth. A perfect match of all nodes and edges with real graph was considered a successful generation.

To assess solely **planning performance** across a broader range of tasks, we conducted experiments on the GraSIF dataset. Since GraSIF contains only graph data, the augmentation module was modified to include nodes connected to the explored nodes. Tokens for graph augmentation were excluded from the TPA calculation, to compare cost of planing exclusively. For evaluation, we used

`gpt-4o-2024-08-06` [27] and Llama3.2-90b-vision [28] as VLM, Llama3.3 and Gemma3 [29] as LM.

Experiments involving GPT-4o are conducted via the OpenAI API. Open source models are executed on two Tesla V100 GPUs, each with 32GB of VRAM. Simulations in VirtualHome and BEHAVIOR-1K are performed with an Nvidia RTX 2080 GPU.

C. Metrics

We use the following four metrics for evaluation. **Success Rate** (**SR**) is the ratio of successfully completed tasks to all tasks. Task is considered successfully completed if all graph nodes are correctly transformed to match their goal configuration. **Average Plan Precision** (**APP**) is the ratio of correctly modified nodes in the generated plan relative to the total number of changed nodes. APP measures the precision of the method in modifying graph nodes to achieve the goal state. **Tokens Per Action** (**TPA**) measures the computational efficiency and cost of a method. It is calculated as the ratio of the total number of tokens used during planning to the number of actions generated.

VII. Results

A. Performance in Dynamic Environment

The results of our evaluation in dynamic environments are summarized in Table II. LookPlanGraph consistently outperforms all baselines in both SR and APP by effectively leveraging visual descriptions and adapting between graph construction and execution. ReAct underperforms due to an overreliance on previous actions and insufficient use of semantic context, often repeating similar searches (e.g., across multiple drawers). In contrast, LookPlanGraph integrates scene descriptions and object priors at every step, enabling more targeted searches and improving SR.

We also observe notable differences between LlamA3.2-90B and GPT-4o. LlamA3.2-90B attains higher SR due to more reliable search behavior, whereas GPT-4o, despite its stronger augmentation capability, performs worse in search by overemphasizing past actions. Static planners fail to surpass 50% SR, revealing fundamental limitations in dynamic scenarios. With ground-truth augmentation, the performance gap between LookPlanGraph and other planners widens further, underscoring the effectiveness of our planning framework.

TABLE II

Methods performance in simulated dynamic environments.

Method	VirtualHome				OmniGibson			
	Llama3.2		GPT-4o		Llama3.2		GPT-4o	
	SR↑	APP↑	SR↑	APP↑	SR↑	APP↑	SR↑	APP↑
LLM-as-P	0.16	0.53	0.44	0.65	0.16	0.36	0.33	0.59
LLM+P	0.00	0.38	0.32	0.58	0.21	0.40	0.37	0.62
SayPlan	0.21	0.53	0.38	0.59	0.10	0.26	0.12	0.31
SayPlan Lite	0.39	0.65	0.48	0.67	0.27	0.55	0.40	0.67
ReAct	0.30	0.64	0.22	0.50	0.33	0.52	0.41	0.64
LookPlanGraph	**0.60**	**0.68**	**0.52**	**0.63**	**0.35**	**0.60**	**0.42**	**0.69**
ReAct$_{GT}$	0.50	0.74	0.34	0.58	0.63	0.75	0.66	0.78
LookPlanGraph$_{GT}$	**0.92**	**0.96**	**0.86**	**0.92**	**0.74**	**0.86**	**0.71**	**0.84**

TABLE III

Graph augmentation capability results.

Model	VirtualHome			OmniGibson		
	F1$_N$	**F1$_E$**	**Success**	**F1$_N$**	**F1$_E$**	**Success**
Llama3.2-90b	0.78	0.73	0.26	0.67	0.67	0.33
GPT-4o	**0.87**	**0.84**	**0.44**	**0.85**	**0.88**	**0.59**

The LLM+P pipeline with LLama3.2-90B exhibits significant challenges with predicate misalignment. Frequent errors include substituting valid predicates with undefined ones (e.g., `on-state` replaced by `on`) and mishandling negations (e.g., `(not (on-state))` replaced by `off`). While increasing the model size has partially alleviated these issues, they remain a persistent problem.

B. Graph Augmentation Capability

Table III presents the results of the graph augmentation module. Using GPT-4o, accurate graphs were generated in 60% of cases across various assets in house and store environments within OmniGibson. Similar performance was observed in VirtualHome, though accuracy decreased with increasing object count: the VLM often underestimates repeated instances (e.g., adding 3 apples instead of the 8 present in the image).

Reducing the size of the model resulted in an overall decrease in recognition capabilities, coupled with problems in producing structured output, reducing the SR to 33%.

C. Planning Performance

Table IV presents the performance of various planning methods across three planning domains in the GraSIF dataset: SayPlan Office, BEHAVIOR-1K, and RobotHow. In the SayPlan Office domain dynamic ReAct approach struggles to recognize correctnes of environment state only from feedback provided on actions frequently making wrong assumtion on task completness. LookPlanGraph surpass this limitation demonstrates the highest SR and APP, even outperforming methods that rely on complete scene graph.

In Behaviour-1k, as the number of changed objects in a task increase, the performance gap between SayPlan Lite and LookPlanGraph narrows as LM struggles to retrieve information about multiple objects states at once, stopping

TABLE IV

Methods comparison on the GraSIF dataset.

Method	SayPlan Office			BEHAVIOR-1K			RobotHow		
	SR↑	APP↑	TPA↓	SR↑	APP↑	TPA↓	SR↑	APP↑	TPA↓
LLM-as-P	0.47	0.59	1409	0.39	0.53	**178**	0.44	0.51	3417
LLM+P	0.07	0.21	1945	0.33	0.37	160	0.30	0.38	5396
SayPlan	0.46	0.59	3697	0.36	0.43	1888	0.86	0.87	5576
SayPlan Lite	0.53	0.68	**1368**	**0.61**	0.76	524	0.84	0.89	4641
ReAct	0.38	0.64	2503	0.47	0.61	1713	**0.89**	**0.91**	**1322**
LookPlanGraph	**0.62**	**0.73**	1989	0.60	**0.77**	1472	0.87	0.89	2653

TABLE V

ABLATION STUDY RESULTS ON THE SAYPLANOFFICE DATASET.

Method / Ablation	SR ↑	APP ↑	TPA ↓
LookPlanGraph	0.62	0.73	1989
Json graph	0.55	0.67	2599
Full Graph	0.65	0.78	4044
No Priors	0.51	0.68	1767
No Correction	0.24	0.40	1682
GPT-4o	0.76	0.83	1752
Llama3.2-90b	0.59	0.72	2004
Gemma3-12b	0.31	0.45	2095

when only part of task executed. Unlike SayPlan Lite, the standard SayPlan approach require access actions, which confuses LM and often resulting in early termination due to feedback interactions limit.

In the RobotHow benchmark, methods using the full scene graph for planning struggled to consistently retrieve key nodes. ReAct, rely on feedback as scene representation, was more effective at accessing relevant information and more token-efficient in dence populated scene. However, it offered only a marginal improvement in 2% over LookPlanGraph, and show limited generalization in more complex environments. These results, along with the observation that different methods outperformed others on TPA metrics depending on the environment, highlight the necessity of evaluation across diverse environments, a capability supported by GraSIF.

D. Ablation study

We analyze key components of LookPlanGraph through ablation studies examining graph representation, system components, and model scalability.

Graph Representation. JSON-based serialization reduces SR and APP while increasing token usage by 30%, showing that natural language descriptions are more effective than structured formats due to absence of explicit format symbols and repeating properties names. Providing to LM full scene graph improves performance but doubles token cost, demonstrating memory graph's efficiency-performance balance.

Component Analysis. Removing object priors slightly reduces SR, confirming that while LookPlanGraph functions without prior knowledge, leveraging available information enhances search effectiveness. The action correction mechanism proves critical, disabling it causes dramatic performance drops, as minor planning errors introduce infeasible actions into resulting plans.

Model Scalability. Performance scales with model capability: GPT-4o achieves highest performance while maintaining efficiency, Llama3.2-90b delivers competitive results with slightly higher token usage, and smaller models like Gemma3-12b show significant degradation but remain functional, which demonstrates LookPlanGraph's adaptability across model sizes.

E. Real-world evaluation

To validate LookPlanGraph's practical applicability, we conducted experiments in a real office environment containing 7 rooms, 75 assets, and 130 objects. We manually constructed a scene graph representing this environment and designed 15 instruction-following tasks targeting commonly moved items such as cups, keys, clothing, and office supplies that require goal understanding and states reasoning.

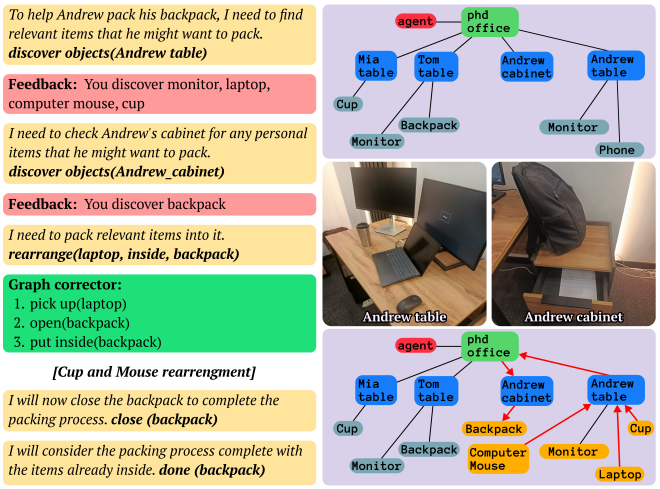

Fig. 4. Execution of "Andrew going home. Help him pack his backpack" task in real environment. Despite the absence of relevant items in the initial graph, LookPlanGraph correctly identified candidate assets, invoked the graph augmentation module to add newly detected objects, and generated a plan to pack a cup, notebook, and mouse into the backpack, while leaving unrelated items untouched. The system subsequently issued a "close backpack" command and marked the task as complete.

Our experimental setup used LookPlanGraph with GPT-4o as the backbone model to analyze the scene graph and generate high-level actions. Following the human-in-the-loop evaluation protocol from Zhang et al. [30], a human operator sequentially executed the generated actions while providing egocentric visual feedback to the graph augmentation module when requested. Figure 4 demonstrates the execution of the task "Andrew going home. Help him pack his backpack".

The system achieved a success rate of 80% (12/15 tasks), demonstrating robust performance in real-world scenarios. Failed cases are analyzed in section VII-F.

F. Failure cases

Our analysis reveals two primary failure modes that limit LookPlanGraph's effectiveness in complex scenarios.

Action Hallucination and Invalid Discovery. The LM occasionally generates semantically invalid actions, such as attempting to "discover" objects that are already known or applying discovery operations to inappropriate node types (e.g., room nodes). These hallucinations contradict the task specifications and cause failures across multiple datasets.

Semantic Knowledge Limitations. The LM struggles with domain-specific reasoning. For example, when instructed to "Clean coffee room after party," it failed to distinguish disposable items (paper cups) from reusable ones (metal cup), leading to inappropriate disposal actions.

VIII. CONCLUSION

We introduce LookPlanGraph, a planning approach that leverages language models as high-level action policies with

efficient memory graph representations. Unlike previous LM-based graph planners, our method operates dynamically, enabling exploration of the environment during task execution and adaptation to changes in object positions. Our results demonstrate that LookPlanGraph performs comparably to conventional full-plan planners in fully determined scenarios while significantly outperforming them in dynamic environments. We also introduce the GraSIF dataset with automated, graph-based evaluation tools for more precise assessment.

While our approach shows promising results for high-level planning, it relies on perfect low-level action execution. In real-world scenarios, robots face challenges such as sensor noise, grasping failures, and navigation errors. Future work should integrate robust error recovery mechanisms and feedback from low-level controllers to enhance system resilience to execution failures. Dynamic planning holds significant potential for real-world robotic systems, enabling seamless human collaboration and more efficient real-time replanning during task execution.

REFERENCES

[1] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[2] 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.

[3] Q. Gu, A. Kuwajerwala, S. Morin, K. M. Jatavallabhula, B. Sen, A. Agarwal, C. Rivera, W. Paul, K. Ellis, R. Chellappa, C. Gan, C. M. de Melo, J. B. Tenenbaum, A. Torralba, F. Shkurti, and L. Paull, "Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning," 2023. [Online]. Available: https://arxiv.org/abs/2309.16650

[4] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. D. Reid, and N. Suenderhauf, "Sayplan: Grounding large language models using 3d scene graphs for scalable task planning," *CoRR*, 2023.

[5] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone, "Llm+p: Empowering large language models with optimal planning proficiency," 2023. [Online]. Available: https://arxiv.org/abs/2304.11477

[6] Y. Liu, L. Palmieri, S. Koch, I. Georgievski, and M. Aiello, "Delta: Decomposed efficient long-term robot task planning using large language models," 2024. [Online]. Available: https://arxiv.org/abs/2404.03275

[7] D. Honerkamp, M. Büchner, F. Despinoy, T. Welschehold, and A. Valada, "Language-grounded dynamic scene graphs for interactive object search with mobile manipulation," *IEEE Robotics and Automation Letters*, 2024.

[8] Y. Tang, M. Wang, Y. Deng, Z. Zheng, J. Deng, and Y. Yue, "Openin: Open-vocabulary instance-oriented navigation in dynamic domestic environments," *arXiv preprint arXiv:2501.04279*, 2025.

[9] C. Li, R. Zhang, J. Wong, C. Gokmen, S. Srivastava, R. Martín-Martín, C. Wang, G. Levine, W. Ai, B. Martinez, *et al.*, "Behavior-1k: A human-centered, embodied ai benchmark with 1,000 everyday activities and realistic simulation," *arXiv preprint arXiv:2403.09227*, 2024.

[10] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba, "Virtualhome: Simulating household activities via programs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8494–8502.

[11] C. Agia, K. M. Jatavallabhula, M. Khodeir, O. Miksik, V. Vineet, M. Mukadam, L. Paull, and F. Shkurti, "Taskography: Evaluating robot task planning over large 3d scene graphs," in *Conference on Robot Learning*. PMLR, 2022, pp. 46–58.

[12] P. Doherty and J. Kvarnstram, "Talplanner: A temporal logic-based planner," *AI Magazine*, vol. 22, no. 3, pp. 95–95, 2001.

[13] Y. Zhang, J. Yang, J. Pan, S. Storks, N. Devraj, Z. Ma, K. P. Yu, Y. Bao, and J. Chai, "Danli: Deliberative agent for following natural language instructions," *arXiv preprint arXiv:2210.12485*, 2022.

[14] K. Zheng, K. Zhou, J. Gu, Y. Fan, J. Wang, Z. Di, X. He, and X. E. Wang, "Jarvis: A neuro-symbolic commonsense reasoning framework for conversational embodied agents," *arXiv preprint arXiv:2208.13266*, 2022.

[15] 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.

[16] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[17] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, *et al.*, "Inner monologue: Embodied reasoning through planning with language models," *arXiv preprint arXiv:2207.05608*, 2022.

[18] Y. Chen, H. Sawhney, N. Gydé, Y. Jian, J. Saunders, P. Vela, and B. Lundell, "A schema-guided reason-while-retrieve framework for reasoning on scene graphs with large-language-models (llms)," *arXiv preprint arXiv:2502.03450*, 2025.

[19] 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.

[20] R. Wang, P. Jansen, M.-A. Côté, and P. Ammanabrolu, "Scienceworld: Is your agent smarter than a 5th grader?" *arXiv preprint arXiv:2203.07540*, 2022.

[21] 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.

[22] M. Booker, G. Byrd, B. Kemp, A. Schmidt, and C. Rivera, "Embodiedrag: Dynamic 3d scene graph retrieval for efficient and scalable robot task planning," *arXiv preprint arXiv:2410.23968*, 2024.

[23] D. Ekpo, M. Levy, S. Suri, C. Huynh, and A. Shrivastava, "Verigraph: Scene graphs for execution verifiable robot planning," *arXiv preprint arXiv:2411.10446*, 2024.

[24] N. Hughes, Y. Chang, and L. Carlone, "Hydra: A real-time spatial perception system for 3d scene graph construction and optimization," 2022. [Online]. Available: https://arxiv.org/abs/2201.13360

[25] J. Wen, Y. Zhu, J. Li, M. Zhu, Z. Tang, K. Wu, Z. Xu, N. Liu, R. Cheng, C. Shen, *et al.*, "Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation," *IEEE Robotics and Automation Letters*, 2025.

[26] M. Shridhar, X. Yuan, M.-A. Côté, Y. Bisk, A. Trischler, and M. Hausknecht, "Alfworld: Aligning text and embodied environments for interactive learning," *arXiv preprint arXiv:2010.03768*, 2020.

[27] 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.

[28] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[29] G. Team, M. Riviere, S. Pathak, P. G. Sessa, C. Hardin, S. Bhupatiraju, L. Hussenot, T. Mesnard, B. Shahriari, A. Ramé, *et al.*, "Gemma 2: Improving open language models at a practical size," *arXiv preprint arXiv:2408.00118*, 2024.

[30] W. Zhang, M. Wang, G. Liu, X. Huixin, Y. Jiang, Y. Shen, G. Hou, Z. Zheng, H. Zhang, X. Li, *et al.*, "Embodied-reasoner: Synergizing visual search, reasoning, and action for embodied interactive tasks," *arXiv preprint arXiv:2503.21696*, 2025.