LookPlanGraph: Embodied instruction following method with VLM graph augmentation

Anatoly O. Onishchenko[1] **Alexey K. Kovalev**[1,2] **Aleksandr I. Panov**[1,2]
[1]Moscow Institute of Physics and Technology, Dolgoprudny, Russia
[2]AIRI, Moscow, Russia
onishchenko.ao@phystech.edu, {kovalev,panov}@airi.net

Abstract

Recently, approaches using Large Language Models (LLM) as planners for robotic tasks have become widespread. In such systems, the LLM must be grounded in the environment in which the robot is operating in order to successfully complete tasks. One way to achieve this grounding is to use a scene graph that contains all the information necessary to complete the task, including the presence and location of objects. In this paper, we propose an approach that works with a scene graph containing only immobile static objects, and augments the scene graph with the necessary movable objects during instruction following using a visual language model and an image from the agent's camera. We conduct thorough experiments on the compiled GRASIF dataset that contain tasks from SayPlan Office, Behaviour-1K, and RobotHow datasets, and demonstrate that the proposed approach effectively handles the task, bypassing approaches that use pre-created scene graphs.

1 Introduction

The pursuit of autonomous agents that can comprehend and execute complex human instructions in dynamic environments is a fundamental objective in robotics. Recent strides in Large Language Models (LLMs) have shown significant potential in reasoning and planning for a variety of tasks articulated in natural language (Huang et al., 2022; Ahn et al., 2022; Kovalev & Panov, 2022; Singh et al., 2023; Sarkisyan et al., 2023). For robots to effectively carry out these tasks, it is crucial that LLMs are grounded in the physical environments where the robots operate. One effective strategy for achieving this grounding is through the use of scene graphs (Gu et al., 2023), which offer structured representations of environments by detailing objects and their interrelationships.

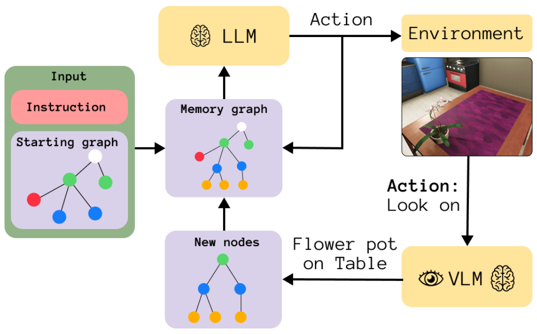

Figure 1: **LookPlanGraph** enhances an agent's ability to operate in dynamic environments by integrating real-time updates from the environment into its graph representation.

Traditionally, the processes of constructing a scene graph and executing tasks using it have been treated separately. SayPlan (Rana et al., 2023) leverages static scene graph representations to generate viable task plans for embodied agents. However, this reliance on static graphs presupposes unchanging environments, a condition rarely met in real-world scenarios where objects frequently change locations or states. Consequently, when the environment undergoes changes, methods such as SayPlan require the entire scene graph to be reconstructed. This reconstruction involves additional procedures like scene navigation, image capturing, and data analysis, all of which are time-intensive and computationally demanding, thus impeding real-time application.

The assumption of a static scene graph is particularly impractical in dynamic settings for several reasons. Firstly, other agents or unforeseen events may alter the state, location, or relationships of objects within the environment. Secondly, certain objects might be concealed within closed containers like boxes or cabinets. Including these hidden objects in the initial graph would necessitate a thorough examination of all possible storage spaces during graph construction, an approach that is neither efficient nor scalable. Therefore, an agent operating in a dynamic environment must possess the capability to dynamically extend and update its scene graph based on real-time observations made during task execution.

Another major limitation of existing methods like SayPlan is its reliance on large, computationally heavy closed models like GPT-4. Although these models are highly effective, their significant resource demands create challenges for applications that requires local computation. To overcome these challenges, we present three key contributions:

1. **SayPlan Lite:** We present SayPlan Lite, a streamlined version of the original SayPlan method, designed to boost the efficiency of smaller LLMs for local machine use. Its success showcases the potential for broader application in resource-limited contexts, making it a viable tool for building LLM agents tailored to constrained environments.
2. **LookPlanGraph:** We propose LookPlanGraph, a graph-based planning framework for dynamic environments. Unlike static scene graphs, this approach initializes with unmovable assets and dynamically updates with movable objects using a Visual Language Model (VLM) and the agent's egocentric camera. It employs a Memory Graph Mechanism to adapt to environmental changes by focusing on relevant, nearby objects, reducing computational demand. A Graph Augmentation Mechanism further allows real-time exploration and updates, ensuring adaptability to the agent's surroundings.
3. **The GRASIF Dataset:** We have created a 558-task dataset GRASIF (Graph Scenes for Instruction Following) for graph-based instruction-following, featuring automated validation. Built from SayPlan Office, Behaviour-1K, and VirtualHome RobotHow environments, this dataset offers a robust resource for assessing planning methods across diverse settings.

2 Related works

2.1 Embodied Planning

Robotic task planning generates sequences of actions to achieve goals within an environment. Traditional methods use domain-specific languages, such as Planning Domain Definition Language (PDDL) (Fox & Long, 2003) and Temporal Logic (TL) (Doherty & Kvarnstram, 2001), combined with parsing, search methods, and heuristics. These are effective in controlled settings but struggle with scalability and generality in complex environments. Recently, LLMs are being utilized for task planning due to their in-context learning abilities. Huang et al. (2022) used LLMs to translate actions into executable commands specific to environments. LOTA-Bench (Choi et al., 2024) employs LLMs to predict the next action based on sequence probability, while LLM+P approach (Liu et al., 2023) integrates grounding by creating a PDDL description for classical planners.

2.2 Planning with graph representation

Effective grounding requires a reliable environmental representation, typically achieved through scene graphs that structure entities and their relationships (Gu et al., 2023; Liu et al., 2021; Devarakonda et al., 2024). However, as scene complexity increases, different methods have been introduced to manage graph size and relevance. For PDDL-based planning, Taskography (Agia et al., 2022) employs a filtering strategy that hides unrelated nodes by matching task-relevant node names with those in the graph. Delta (Liu et al., 2024b) extends the LLM+P approach by using scene graphs to define PDDL domains, leveraging LLMs to prune irrelevant nodes. SayPlan (Rana et al., 2023) follows a similar strategy, using semantic search to iteratively reveal and hide nodes from different rooms until all task-relevant objects are found, with an LLM guiding the process. Additionally, Chen et al. (2025) introduces a retriever module that dynamically generates code to answer graph-related queries from the planner during task execution. While these methods effectively reduce graph complexity and optimize context length for LLMs, they are not well-suited for dynamic environments, due to unreliability provided in graph object positions.

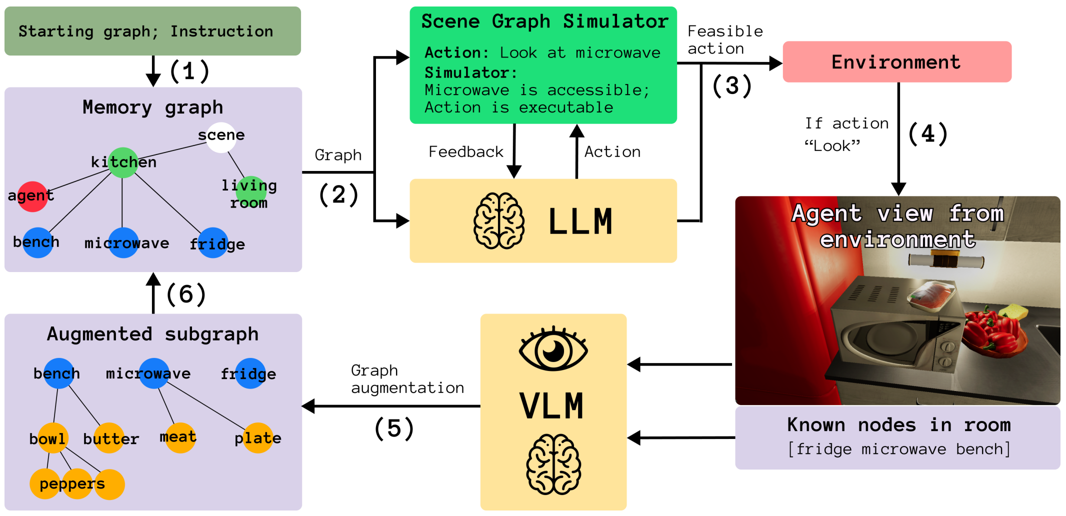

Figure 2: **LookPlanGraph Overview:** The LookPlanGraph starts with an instruction and a static environment graph **(1)**. A memory graph, initially a copy of the starting graph, is processed by the LLM with the task description and is also sent to the Scene Graph Simulator **(2)**. The LLM suggests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

2.3 Dynamic graph augmentation

ConceptGraphs (Gu et al., 2023) constructs 3D scene graphs using pre-collected images of the environment, allowing graph augmentation by incorporating more recent visual data. OpenIN (Tang et al., 2025) dynamically builds graphs by segmenting a robot's input during navigation tasks. However, both methods struggle with occluded objects stored inside closed containers, as the agent lacks the ability to explore their interiors. Moma-LLM (Honerkamp et al., 2024) addresses this limitation by introducing an "open" action, allowing the agent to examine occluded objects. However, this approach depends on ground-truth semantic masks and object localization rather than constructing graphs purely from real-time observations. VeriGraph (Ekpo et al., 2024) leverages VLMs to construct task-specific graph representations and generate plans based on them. While VLMs demonstrate promising capabilities in graph construction, the evaluation of VeriGraph remains restricted to tabletop environments, limiting its applicability to more complex, dynamic settings.

3 Problem formulation

We address the challenge of enabling an autonomous mobile manipulator robot to plan, navigate, and manipulate objects within large-scale household environments using natural language instructions. The robot must reason about dynamic scenes and adapt to environmental changes, such as object locations and states. Our solution necessitates generating executable actions that involve complex navigation and manipulation tasks, effectively accommodating the dynamics of multi-room environments where static representations are inadequate.

Graphs offer a layered representation of the environment, integrating spatial semantics with the relationships between objects while also capturing key states, and properties associated with entities in the scene. In the case of a 3DSG, this is structured as a hierarchical multigraph $G = (V, E)$. The vertex set V is organized into several hierarchical levels: $V_1 \cup V_2 \cup \cdots \cup V_K$. Each group V_k consists of vertices at a specific hierarchical level k. The edges emanating from any vertex $v \in V_k$

are restricted to connect to vertices in $V_{k-1} \cup V_k \cup V_{k+1}$, thereby ensuring that connections occur within the same level or to adjacent levels.

Formally, given a 3DSG G and a task instruction T expressed in natural language, our framework, LookGraphPlan, can be conceptualized as a high-level decision-making module denoted by $\pi(\mathbf{a} \mid T, G)$. This decision-making module is capable of generating an action $\mathbf{a}$ that is grounded in the environment where the embodied agent operates. Furthermore, the generated action is designed to concurrently follow the instruction to be carried out.

Our approach focuses on two primary challenges: 1) Develop a framework capable of operating with scene representations that initially include only the static elements of the environment. This framework must enable the agent to iteratively expand and modify the scene graph during task execution, reflecting changes in the environment; 2) Enabling effective planning using smaller LLMs, that can run locally, to reduce reliance on computationally intensive, large-scale models.

4 METHOD

We introduce LookPlanGraph, a scalable method leveraging scene graphs to operate in environments without predefined states or positions for movable objects. Central to this approach is the Scene Memory Graph (SMG), which is continuously maintained and updated to reflect the current state of the environment. This method empowers exploration and interaction within the environment, augmenting the SMG with newly detected objects.

Illustrated in Figure 2, the LookPlanGraph methodology integrates an LLM, a scene graph representation, and VLM to execute tasks. The process begins with an starting graph outlining the rooms and fixed assets within the environment, which is then replicated into the SMG for use throughout the method. As the agent engages with the environment, the SMG is dynamically updated, incorporating modifications made by the agent via the Scene Graph Simulator and new objects identified by the VLM. Algorithm 1 of the method follows a structured cycle (4-15):

Algorithm 1 LookPlanGraph

```
 1: Given: LLM planner LLM, VLM parser VLM, Environment ENV, Memory graph M, Graph Simulator $Sim$
 2: Inputs: Starting graph G, Task T
 3: $M = G$
 4: while action ! = done do
 5:     while feedback ! = None do
 6:         action ← $LLM(M, T, feedback)$
 7:         feedback ← $Sim(M, action)$
 8:     end while
 9:     ENV(action)
10:     if action=$'look\_on'$ then
11:         new nodes ← VLM$(ENV, M)$
12:         $M$.append(new nodes)
13:     end if
14: end while
```

LLM Decision-Making (7): The SMG is encoded into a prompt and provided to the LLM, along with the task instructions. Using this input, the LLM determines the next action for the agent to perform. In our approach, the list of possible actions is limited to manipulation tasks such as *goto, pick up, open, close, put on, put in,* and two scene exploration actions: *look on* and *look inside*.

Simulation and Feedback (5-9): The proposed action is sent to the Scene Graph Simulator, which evaluates its feasibility. If the action is valid, the simulator updates the SMG to reflect the outcome. If the action is invalid, feedback is returned to the LLM for re-planning.

Environment Interaction (10): Once validated by the simulator, the action is executed in the real environment.

Graph Augmentation via VLM (11-14): For exploration actions, such as *look on* or *look inside*, the VLM is invoked. The VLM processes images of the environment and generates nodes for newly identified objects. These objects are then added to the SMG.

4.1 MEMORY GRAPH

The Memory Graph (Figure 3) is a hierarchical, graph-based structure inspired by 3D Scene Graphs (Kim et al., 2019; Kurenkov et al., 2021). It encodes spatial semantics, object relationships, and states for efficient robotic planning (Gay et al., 2019; Rosinol et al., 2021). Organized into four layers—Scene, Place, Asset, and Object—it abstracts the environment at different levels. The Scene

Layer represents the entire environment, the Place Layer defines areas (e.g., rooms), the Asset Layer includes immovable objects, and the Object Layer contains movable items. An additional agent node tracks the agent's position and interactions.

Each node in the graph has specific attributes based on its type. These attributes provide detailed information that can be used in the planning and task execution process:

- **Scene Node:** Contains the name of the scene.
- **Place Node:** Contains the scene it belongs.
- **Asset Node:** Includes the asset's place, its state, and properties.
- **Object Node:** Describes the object's state and properties, including the name of the node to which the object belongs and the type of relationship.
- **Agent Node:** Tracks the location of the agent and the item currently held.

Example of text serialized graph shown in Appendix A.

4.2 Action generation

Action generation by the LLM involves structured prompting, graph filtering, and a feedback loop with the Scene Simulator. The prompt consists of three parts: a static prompt, dynamic components, and feedback. The static prompt describes actions and states derived from the graph, focusing on essential home environment tasks like pick-and-place, opening/closing, and turning devices on/off. Additional actions, such as "look on" and "look inside", enable visual interaction. A detailed prompt structure and the full action list are provided in Appendix A.

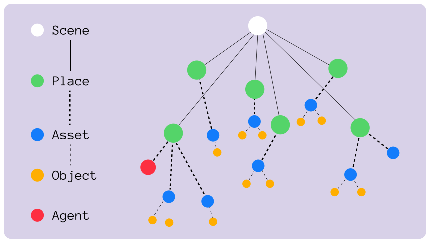

Figure 3: Memory graph structure constructed with four layers and an additional agent node to track the agent.

To optimize for compact LLM models, the prompt is concise yet informative by including only objects in the agent's immediate environment (e.g., the same room) and adding previously interacted objects, especially for long-term planning tasks. The Scene Graph Simulator, similar to SayPlan, ensures actions are executable in the real environment. It attempts to execute LLM-generated actions and updates the memory graph and environment state if successful. If an action fails due to constraints, the simulator provides feedback, which is included in the next prompt to improve subsequent actions.

4.3 Graph augmentation

For visual interaction with the environment, the LLM can inspect the top or interior of accessible assets by calling corresponding action and incorporate newly discovered objects into the memory graph. Example of such interaction shown in the right part of Figure 2. This process involves capturing an image from the environment that represents the agent's field of view. The image is then passed to a Vision-Language Model along with list of assets and objects already present in the memory graph to ensure that only new nodes are added. The VLM is prompted to identify new objects, their states, and their relationships with existing assets. Subsequently, the identified nodes are integrated into the memory graph, making them available for future interactions.

5 The GRASIF dataset

Graph-based scene representation methods are gaining traction in research, yet there is a noticeable lack of 3DSG datasets tied to specific tasks, which has led researchers to rely on proprietary data. To bridge this gap, we have curated a comprehensive GRASIF dataset by integrating resources from multiple sources. Specifically, we combined textual instructions and environmental data from Behaviour-1K (Li et al., 2024), VirtualHome RobotHow (Puig et al., 2018), and SayPlan Office (Rana et al., 2023). While Behaviour-1K lacks task descriptions and graph representations, VirtualHome RobotHow offers graph representations but not in the 3DSG format, and SayPlan Office lacks a coded implementation. Our curated dataset addresses these limitations, providing a valuable resource for evaluating and advancing graph-based methods in robotics research.

We constructed initial and goal graphs for each task, representing environment states before and after execution. This approach enables automatic validation across large, complex environments, reducing reliance on human evaluation. Combined, the datasets cover 10 environments and 558 tasks paired with initial and goal graphs, highlighting various aspects of embodied planning. Dataset characteristics are summarized in Table 1.

Table 1: Comparison of datasets: SayPlan Office is the largest environment, Behaviour-1K offers the most long-horizon tasks, and VirtualHome features numerous objects in compact environments.

Dataset	**Tasks**	**Rooms**	**Nodes**	**Changed nodes**
SayPlan Office	25	37	202.6	2.1
Behaviour-1k	186	1.23	12.1	4.9
VirtualHome	347	4	195.7	1.6

SayPlan Office. For evaluating our method on diverse, human-formulated tasks, we used the SayPlan Office Dataset. As the original dataset is unavailable, we reconstructed environment graph representations based on details from the original paper. To align the graph format with our 3DSG structure, we removed pose nodes and connected rooms directly to the scene node. Graph representations and corresponding action sequences were manually constructed, selecting tasks from simple and complex planning sections of the paper. Using the reconstructed initial graphs and action sequences, we utilized a Scene Graph Simulator to generate goal graph representations after task execution. Ambiguous tasks, like ”Put an object into a place where I can enjoy it,” were excluded. The final dataset consists of 25 tasks, each with instructions and pairs of initial and goal graphs, as detailed in Appendix C.

BEHAVIOR-1K. The Behaviour-1k dataset includes descriptions of 1,000 tasks relevant to real-world scenarios, paired with a simulator providing rooms, scenes, and PDDL task descriptions. We construct graph representations of the environment from PDDL descriptions. The initial graph is derived using a rule-based approach with ontop and inroom predicates. The goal state is generated by GPT-4o (Achiam et al., 2023), which modifies the initial graph based on the task goal and provides human-like task instructions and step-by-step plans. To focus on manipulation tasks, we filter the Behaviour-1k dataset to exclude tasks requiring cooking or cleaning skills, selecting only tasks with predicates `ontop`, `real`, `inside`, `open`, and `toggled on`. This results in 186 tasks with initial and goal graph pairs. The graphs are constructed solely from task descriptions, excluding nodes unrelated to instruction following. This allows for a clear evaluation of planning performance without the need to filter out irrelevant nodes.

VirtualHome RobotHow. RobotHow dataset (Liao et al., 2019), designed for VirtualHome, which includes 1,800 tasks with initial and goal state graphs across 7 home environments. Tasks are filtered to focus on robot-performable manipulation actions, excluding irrelevant tasks (e.g., “Play video game”, “Get shower”) and those involving doors, as they are not represented in the graph structure. Scene descriptions in VirtualHome are translated into graphs using a rule-based approach. Non-grabbable objects are treated as asset nodes, and redundant nodes (e.g., multiple floors, ceilings, walls) are removed for simplicity. The final dataset includes 347 tasks, with duplicates across environments noted. Details on actions and filtered tasks are in Appendix C.

6 EXPERIMENTS

6.1 BASELINES

We evaluate LookPlanGraph against two baseline methods that incorporate LLM as graph planners.

SayPlan (Rana et al., 2023), operates in two distinct stages: semantic search and iterative replanning. During the semantic search stage, the LLM identifies a minimal sufficient scene graph by expanding room nodes containing relevant items. In the iterative replanning stage, the graph is used to query the same LLM for generating a high-level plan, which is revised based on graph simulation feedback. Notably, both stages are executed sequentially using the same LLM dialogue, with the same prompt being called for both operations.

Since the open-source implementation of SayPlan is unavailable, we developed our own version, referred to as SayPlan*. This version is adapted to process 3DSGs without pose nodes. Additionally, as the graph representation in our context does not require the access functions used for real robots in the original paper, we replaced the access and release function combination with simpler "put on" and "put in" functions.

To reduce the complexity of the task for the LLM and enhance efficiency, we introduced a simplified variant of SayPlan, named **SayPlan Lite**. This approach decomposes the planning process into two distinct LLM dialogues: one dedicated to semantic search and the other to iterative replanning. By separating these tasks and few-shot learning examples, we streamline the planning for LLM, making it more manageable and effective, especially for smaller models. Additionally, we refined the representation of the graph within the prompts to further simplify communication with the LLM. A detailed explanation of these modifications can be found in Appendix A.

6.2 Experimental setup

Our experimental framework rigorously evaluates the model's performance in generating plans in a variety of environments represented in our dataset. Each task in the experimental framework consists of a task description, an initial environment graph, and corresponding environment functions. The models iteratively process these inputs to propose plans or determine actions, which are then executed within a scene graph simulator. The resulting graph is then compared to a predefined goal graph, allowing the computation of various performance metrics (Section 6.3).

To compare **planning performance**, experiments are conducted on the GRASIF dataset using a scene graph-based simulator. Given the graph structure of GRASIF, the LookPlanGraph augmentation mechanism is modified to add nodes that connected to explored nodes. For all methods replanning process is uniformly constrained to a maximum of five iterations.

The graph augmentation capability was evaluated using tasks from the Behaviour-1K framework. Images of required assets were collected from the OmniGibson simulator. Accuracy was measured using the F1-score, comparing the generated nodes and edges to the ground truth scene graph. A perfect match of both nodes and edges was considered a successful generation.

In these experiments, models were utilized, including Llama3.3 (Dubey et al., 2024) with 70 billion parameters, Gemma2 (Team et al., 2024) with 27 billion parameters, and gpt-4o-2024-08-06 (Achiam et al., 2023) for planning tasks. For visual language tasks gpt-4o-2024-08-06, Llama3.2 (Dubey et al., 2024) with 90 billion parameters and LLaVa (Liu et al., 2024a) with 34 billion parameters were employed. Local models was running on a server equipped with two Tesla V100 GPUs, each with 32GB of VRAM, while other model experiments were conducted via the official OpenAI API.

6.3 Metrics

We use the following four metrics to evaluate the performance of methods based on a scene graph.

Success Rate (**SR**) is the ratio of successfully completed tasks to all tasks: SR $= \frac{S}{N}$, where S – the number of successful tasks, N – the number of tasks. A 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 modified nodes: APP $= \frac{1}{N}\sum_{i=1}^{N}\frac{M_i^C}{M_i}$, where N – the number of tasks, M_i^C – the number of correctly modified nodes in task i, M_i – the number of modified nodes in task i. APP measures the precision of the method in modifying graph nodes to achieve the goal state.

Average Plan Length (**APL**) is the number of actions required to achieve the goal state, averaged over successfully completed tasks: APL $= \frac{1}{S}\sum_{i=1}^{S} L_i$, where S – the number of successful tasks, L_i – the plan length in task i. APL evaluates the efficiency of the generated plans, with shorter plans generally being preferred, provided they achieve task success.

Node Relevance Ratio (**NRR**) is the ratio of observed nodes to the number of important nodes, where important nodes are those that differ between the initial and goal graphs: NRR $= \frac{1}{N}\sum_{i=1}^{N}\frac{K_i^O}{K_i^I}$. where N – the number of tasks, K_i^O – the number of observed nodes in task i, K_i^I – the number of important nodes in task i. NRR measures the ability of the method to focus on task-relevant nodes during the planning process. A lower NRR indicates that the method observes fewer unnecessary nodes, which helps reduce the number of tokens used and improves computational efficiency.

7 Results

The evaluation is divided into three key areas, each focusing on a specific aspect. First, we assess the planning performance of various methods on the GRASIF dataset. Second, we analyze the impact of language model size on method performance, highlighting its influence on plan precision. Finally, we evaluate the graph augmentation capabilities of LookPlanGraph using different visual models.

7.1 Planning capability

Table 2 presents the performance of various methods utilizing the Llama3.3 model for planning across three datasets within the GRASIF framework. The integration of Llama3.3 within the SayPlan pipeline encounters challenges during the semantic search stage owing to erroneous API calls. The model frequently neglects the initial instructions, attempting to explore non-target nodes or execute actions prematurely intended for the iterative replanning phase. This issue is effectively resolved with SayPlan Lite, which partitions these stages into two distinct dialogues.

Table 2: Methods comparison for Llama3.3.

Method	SR	APP	APL	NRR
SayPlan Office				
SayPlan*	0.00	0.00	0.00	-
SayPlan Lite	**0.16**	**0.39**	**4.04**	19.83
LookPlanGraph	0.12	0.34	6.01	**15.72**
BEHAVIOR-1K				
SayPlan*	0.00	0.00	0.00	-
SayPlan Lite	**0.56**	**0.65**	11.56	3.04
LookPlanGraph	0.33	0.41	**10.53**	**2.67**
RobotHow				
SayPlan*	0.00	0.00	0.00	-
SayPlan Lite	**0.39**	**0.41**	**2.08**	40.51
LookPlanGraph	0.25	0.26	3.14	**2.67**

LookPlanGraph shows slightly lower overall performance, but it achieves a higher Node Relevance Ratio. This indicates a more efficient Memory graph filtering strategy, which also allows the agent to operate effectively in a dynamic environment.

Both approaches demonstrate superior performance when applied to Behaviour-1k tasks, which benefit from pre filtered graphs, underscoring the crucial role of the filtering phase. In the RobotHow dataset, a primary issue for LookPlanGraph was the inappropriate utilization of exploratory extensions. For example, in "turn on lights" tasks, the agent incorrectly inspected the room lighting and executed the "done" action without activating the essential switch.

7.2 Impact of LLM on planning performance

The precision of planning for various LLMs is shown in Table 3. The results demonstrate a dependency of planning methods on the size and capabilities of the LLMs used. The SayPlan* method achieves an precision of 0.48 with the GPT-4o model, while it is ineffective with smaller models such as Llama3.3-70b and Gemma 27b. This reduced performance in smaller models is attributed to an increased susceptibility to hallucinations, resulting in unreliable function calls and node predictions. Although GPT-4o also experiences hallucinations, they occur less frequently due to its larger parameter space and enhanced contextual understanding. The discrepancy from the orig-

Table 3: Average Plan Precision for different models on the SayPlan Office dataset.

Method	GPT-4o	Llama3.3	Gemma2
SayPlan*	0.48	0.00	0.00
SayPlan Lite	0.61	**0.39**	0.00
LookPlanGraph	**0.63**	0.34	**0.15**

inal study's reported 80% success rates on SayPlanOffice is likely due to differences in evaluation metrics; the original study's inclusion of human evaluation may have allowed for more ambiguous solutions to be deemed successful.

SayPlan Lite exhibits balanced accuracy, especially with the Llama3.3-70b model, achieving 0.39 precision. Its modular prompt structure helps to reduce planning inaccuracies in smaller models. The LookPlanGraph method attains marginally higher precision, with values of 0.63 on the GPT-4o model and 0.15 on Gemma2 models. This is attributed to its ability to dynamically iterate actions, allowing the agent to complete portions of tasks before encountering incomprehensible state.

7.3 Graph augmentation capability

The results of the graph augmentation pipeline are presented in Table 4. Graph augmentation using GPT-4o, successfully generated accurate graphs in 60% of instances across diverse assets in house and store environments within the OmniGibson simulator. However, we found that the VLM struggled when presented with a large number of objects, typically around ten. In scenarios with fewer objects, the VLM could successfully identify them, but sometimes encountered challenges with naming – for example, mislabeling a candle as a cylinder, which does not count as accurate recognition.

Table 4: Graph augmentation capability results.

	F1 Score		
Method	**Node**	**Edge**	**Success**
LLaVa	-	-	-
Llama3.2-90b	0.67	0.67	0.33
GPT-4o	**0.85**	**0.88**	**0.59**

Reducing the size of the model (Llama3.2) resulted in an overall decrease in recognition capabilities, coupled with problems in producing structured output, reducing the SR to 33%. Meanwhile, the LLaVa model was able to identify objects in the provided images, but faced problems with edge detection and producing structured output, resulting in an SR of 0%.

8 Conclusion

In this work, we propose LookPlanGraph, a graph-based planning framework for dynamic environments. LookPlanGraph overcomes the limitations of static graph representations by integrating LLM and VLM to enable real-time updates of the scene graph. While the overall success rate of our method is sometimes lower than that of the lighter version of SayPlan, our experiments with graph augmentation highlight LookPlanGraph's ability to effectively adapt to dynamic environments through active exploration during task execution. In addition, we developed SayPlan Lite, a modular version of SayPlan tailored for smaller LLMs, which reduces the incidence of erroneous function calls. Experimental results consistently show that SayPlan Lite outperforms SayPlan* when applied to smaller models such as Llama3.3, primarily due to its decentralized dialog structure, which enhances protection against erroneous semantic calls. By advancing adaptive, context-aware planning, LookPlanGraph brings LLM-based planning systems closer to real-world applicability. It lays a strong foundation for future breakthroughs in autonomous robotics and interactive agents, paving the way for robust and versatile solutions.

9 Limitations

A fundamental aspect of LookPlanGraph is the construction of a graph representation of the scene using a 3D scene graph structure. This graph-based approach organizes spatial relations and object states, but limits the applicability to environments that fit this model. For example, the current representation mainly supports spatial relationships such as "inside" and "on top", which may not capture the full range of relationships in diverse datasets or real-world scenarios. LookPlanGraph also suffers from the limitations of LLM and VLM, including bias, inaccuracy, and incomplete visual input. These affect decision making, especially for complex tasks. Future improvements such as fine-tuning, better visual models, or alternative sensory inputs could improve reliability. LookPlanGraph assumes perfect low-level action policies, which remains a challenge in robotics. While this simplifies high-level planning, it ignores execution errors, sensor noise, or partial observability. Addressing these challenges would require robust error recovery mechanisms and adaptive control strategies to bridge the gap between high-level plans and real-world execution. Finally, the feedback quality of the scene graph simulator may degrade as task complexity increases, especially with diverse actions and predicates. The development of a more advanced feedback system with improved error detection and correction dialog is a valuable future direction.

10 Ethical Considerations

Our approach is based on a large language model that operates in generation mode, and despite the use of a prompt that limits the output format, the model can potentially generate inappropriate and/or offensive output. In addition, language models are prone to hallucinations and can generally produce unforeseen results, so giving them control over mechanisms that could potentially cause harm and testing such mechanisms should be done in a regulated manner, in a specially designated area with limited access to the people involved in the experiments. It is also potentially possible to deliberately execute harmful plans on a robot with the intent to cause harm.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Christopher Agia, Krishna Murthy Jatavallabhula, Mohamed Khodeir, Ondrej Miksik, Vibhav Vineet, Mustafa Mukadam, Liam Paull, and Florian Shkurti. Taskography: Evaluating robot task planning over large 3d scene graphs. In *Conference on Robot Learning*, pp. 46–58. PMLR, 2022.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Yiye Chen, Harpreet Sawhney, Nicholas Gydé, Yanan Jian, Jack Saunders, Patricio Vela, and Ben Lundell. A schema-guided reason-while-retrieve framework for reasoning on scene graphs with large-language-models (llms). *arXiv preprint arXiv:2502.03450*, 2025.

Jae-Woo Choi, Youngwoo Yoon, Hyobin Ong, Jaehong Kim, and Minsu Jang. Lota-bench: Benchmarking language-oriented task planners for embodied agents. *arXiv preprint arXiv:2402.08178*, 2024.

Venkata Naren Devarakonda, Raktim Gautam Goswami, Ali Umut Kaypak, Naman Patel, Rooholla Khorrambakht, Prashanth Krishnamurthy, and Farshad Khorrami. Orionnav: Online planning for robot autonomy with context-aware llm and open-vocabulary semantic scene graphs, 2024. URL `https://arxiv.org/abs/2410.06239`.

Patrick Doherty and Jonas Kvarnstram. Talplanner: A temporal logic-based planner. *AI Magazine*, 22(3):95–95, 2001.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daniel Ekpo, Mara Levy, Saksham Suri, Chuong Huynh, and Abhinav Shrivastava. Verigraph: Scene graphs for execution verifiable robot planning. *arXiv preprint arXiv:2411.10446*, 2024.

Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.

Paul Gay, James Stuart, and Alessio Del Bue. Visual graphs from motion (vgfm): Scene understanding with object geometry reasoning. In *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part III 14*, pp. 330–346. Springer, 2019.

Qiao Gu, Alihusein Kuwajerwala, Sacha Morin, Krishna Murthy Jatavallabhula, Bipasha Sen, Aditya Agarwal, Corban Rivera, William Paul, Kirsty Ellis, Rama Chellappa, Chuang Gan, Celso Miguel de Melo, Joshua B. Tenenbaum, Antonio Torralba, Florian Shkurti, and Liam Paull. Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning, 2023. URL `https://arxiv.org/abs/2309.16650`.

Daniel Honerkamp, Martin Büchner, Fabien Despinoy, Tim Welschehold, and Abhinav Valada. Language-grounded dynamic scene graphs for interactive object search with mobile manipulation. *IEEE Robotics and Automation Letters*, 2024.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pp. 9118–9147. PMLR, 2022.

Ue-Hwan Kim, Jin-Man Park, Taek-Jin Song, and Jong-Hwan Kim. 3-d scene graph: A sparse and semantic representation of physical environments for intelligent agents. *IEEE transactions on cybernetics*, 50(12):4921–4933, 2019.

Aleksei Konstantinovich Kovalev and Aleksandr Igorevich Panov. Application of pretrained large language models in embodied artificial intelligence. In *Doklady Mathematics*, volume 106, pp. S85–S90. Springer, 2022.

Andrey Kurenkov, Roberto Martín-Martín, Jeff Ichnowski, Ken Goldberg, and Silvio Savarese. Semantic and geometric modeling with neural message passing in 3d scene graphs for hierarchical mechanical search. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11227–11233. IEEE, 2021.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Wensi Ai, Benjamin 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.

Yuan-Hong Liao, Xavier Puig, Marko Boben, Antonio Torralba, and Sanja Fidler. Synthesizing environment-aware activities via activity sketches. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6284–6292, 2019. doi: 10.1109/CVPR.2019.00645.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+p: Empowering large language models with optimal planning proficiency, 2023. URL `https://arxiv.org/abs/2304.11477`.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024a.

Hengyue Liu, Ning Yan, Masood Mortazavi, and Bir Bhanu. Fully convolutional scene graph generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11546–11556, 2021.

Yuchen Liu, Luigi Palmieri, Sebastian Koch, Ilche Georgievski, and Marco Aiello. Delta: Decomposed efficient long-term robot task planning using large language models, 2024b. URL `https://arxiv.org/abs/2404.03275`.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8494–8502, 2018.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian D Reid, and Niko Suenderhauf. Sayplan: Grounding large language models using 3d scene graphs for scalable task planning. *CoRR*, 2023.

Antoni Rosinol, Andrew Violette, Marcus Abate, Nathan Hughes, Yun Chang, Jingnan Shi, Arjun Gupta, and Luca Carlone. Kimera: From slam to spatial perception with 3d dynamic scene graphs. *The International Journal of Robotics Research*, 40(12-14):1510–1546, 2021.

Christina Sarkisyan, Alexandr Korchemnyi, Alexey K Kovalev, and Aleksandr I Panov. Evaluation of pretrained large language models in embodied planning tasks. In *International Conference on Artificial General Intelligence*, pp. 222–232. Springer, 2023.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530. IEEE, 2023.

Yujie Tang, Meiling Wang, Yinan Deng, Zibo Zheng, Jingchuan Deng, and Yufeng Yue. Openin: Open-vocabulary instance-oriented navigation in dynamic domestic environments. *arXiv preprint arXiv:2501.04279*, 2025.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

A APPENDIX – PROMPT STRUCTURES

A.1 LOOKPLANGRAPH

The static prompt remains constant across all tasks and provides foundational information to the LLM. It includes the agent's role and objectives, a description of states and relationships that can appear in the JSON graph representation, a list of functions available to the agent (e.g., ”look on,” ”look inside,” ”pick up”), the expected output format (structured JSON response detailing the next action), and two examples of how the agent should respond.

After the static prompt, dynamic components follow. These include the instruction, which is a natural language description of the task, a filtered JSON graph representation, and feedback. The JSON graph is simplified to include only the nodes and attributes relevant to performing the action, such as those in the same room as the agent or objects the agent interacted with earlier in the task. This filtering ensures the prompt remains concise while providing necessary context for long-horizon tasks.

A.1.1 STATIC PROMPT

Agent Role: You are an expert in graph-based task planning. Given a graph representation of the environment, your goal is to generate a next move for the agent to follow to solve the given instruction.

Graph environment states:

- `ontop_of(<asset>)`: Object is located on `<asset>`.
- `inside_of(<asset>)`: Object is located inside `<asset>`.
- `closed`: Asset can be opened.
- `open`: Asset can be closed or kept open.
- `on`: Asset is currently on.
- `off`: Asset is currently off.

Available Functions:

- `go_to(<room>)`: Move the agent to room node. Use it only with room nodes.
- `pick_up(<object>)`: Pick up an accessible object from the accessed node. You can handle only one item.
- `put_on(<asset>)`: Put held object on `<asset>`.
- `put_inside(<asset>)`: Put held object inside of `<asset>`.
- `turn_on/off(<node>)`: Toggle object on or off.
- `open/close(<node>)`: Open or close node.
- `look_on(<asset>)`: Look on top of `<asset>`. Adds the discovered objects to the memory graph.
- `look_inside(<asset>)`: Look inside of `<asset>`. Adds the discovered objects to the memory graph.
- `done(<node>)`: Call this function with any node when the goal has been achieved.

Answer only with JSON without comments. Output Response Format:

```
{
  "chain_of_thought": Break down your reasoning into intermediate steps.
  "next_move": {
    "function_name": Name of the function from Available Functions.
    "function_target": Node name.
  }
}
```

Examples of output:

```
{
  "chain_of_thought": [
    "i have found the coffee mug,
    the coffee machine and tom’s wardrobe on the graph",
    "collect coffee mug",
    "generate plan for making coffee",
    "place coffee mug on Tom’s wardrobe"
  ],
  "next_move": {
    "function_name": "go_to",
    "function_target": "bobs_room"
  }
}

{
  "chain_of_thought": [
    "goal is reached",
    "i am inside bobs_room",
    "now i call function to show thats i am done with task"
  ],
  "next_move": {
    "function_name": "done",
    "function_target": "bobs_room"
  }
}
```

A.1.2 Dynamic prompt example

Instruction: Take the socks, bottle of perfume, toothbrush, and notebook out of the carton and place them on the sofa in the living room.

Memory graph:

```
{"nodes":{"room":[
{"id":"living_room1"}],
"asset":[{"id":"floor1","located":"living_room1","states":[]},
{"id":"sofa1","located":"living_room1","states":[]}],"object":[
{"id":"carton1","relation":"ontop_of","related_to":"sofa1","states":["closed"]},
{"id":"sock1","relation":"ontop_of","related_to":"sofa1","states":[]},
{"id":"sock2","relation":"ontop_of",
"related_to":"sofa1","states":[]},
{"id":"bottle__of__perfume1","relation":"ontop_of",
"related_to":"sofa1","states":["closed"]},
{"id":"toothbrush1","relation":"ontop_of","related_to":"sofa1","states":[]},
{"id":"notebook1","relation":"ontop_of","related_to":"sofa1","states":[]}],
"agent":[{"id":"agent1","location":"living_room1","holding":""}]}}
```

A.2 SayPlan Lite

SayPlan Lite splits the prompt into two stages corresponding to SayPlan’s workflow, hiding irrelevant information at each stage and separating the LLM’s API interactions into two parts. This approach minimizes potential hallucinations.

A.2.1 SEMANTIC SEARCH

```
Agent Role:
You are an efficient graph search agent tasked with exploring
a graph-based environment to  find specific items based on a given instruction.
You interact with the environment via an API to expand or contract room nodes.
Objective:
Your goal is to identify the relevant
parts of the graph to fulfill the instruction.
You must expand appropriate room nodes, filter out irrelevant ones,
and verify the graph using the environment's API.

Environment API:
expand_node(<room>): Reveal assets/objects connected to a room node.
contract_node(<room>): Hide assets/objects, reducing
graph size for memory constraints.
verify_plan(): Verify graph in the scene graph environment.

Guidelines:
1. Do not expand asset or object nodes, only room nodes.
2. Contract irrelevant nodes to reduce memory usage.
3. Once all relevant objects are found, use verify_plan() to confirm that graph
is rellevant to the task.

Output Response Format: Your response should follow this structure:
{
"chain_of_thought": break your problem down into a series of intermediate
reasoning steps to help you determine your next command,
"reasoning": justify why the next action is important
"command":
  {
  "command_name": Environment API call
  "node_name": node to perform an operation on
  }
}

Example of output:
{
  "chain_of_thought": [
    "i have found a wardrobe in tom's room",
    "leave this node expanded",
    "the coffee mug is not in his room",
    "still have not found the coffee machine",
    "kitchen might have coffee machine and coffee mug",
    "explore this node next"
  ],
  "reasoning": "i will expand the kitchen next",
  "command": {
    "command_name": "expand_node",
    "node_name": "kitchen1"
  }
}
```

A.2.2 ITERATIVE RE-PLANNING

```
Agent Role: You are an expert in graph-based task planning.
Given a graph representation of the environment,
your goal is to generate a precise, step-by-step task plan
```

```
for the agent to follow and solve the given instruction.

Graph environment states:
ontop_of(<asset>): Object is located on <asset>
inside_of(<asset>): Object is located inside <asset>
attached_to(<asset>): Object is attached to <asset>
closed: Asset can be opened
open: Asset can be closed or kept open
on: Asset is currently on
off: Asset is currently off

Available Functions (use these exclusively for planning):
go_to(<room>): Move the agent to room node. Use it only with room nodes.
pick_up(<object>): Pick up an accessible object from the accessed node.
You can handle only one item.
put_on(<asset>): Put holded object on asset.
put_inside(<asset>): Put holded object inside of asset.
put_under(<asset>): Put holded object under of asset.
attach(<asset>): Attach holded object to asset.
turn_on/off(<object>): Toggle object at agent’s node,
if accessible and has affordance.
open/close(<node>): Open/close node at agent’s node, affecting object.

Answer only with JSON without comments. Output Response Format:
{"chain_of_thought": Break down your reasoning into intermediate steps.
"plan": List the environment function calls to solve the task.}

Example of output:
{
  "chain-of-thought": [
    "i have found the coffee mug,
    the coffee machine and tom’s wardrobe on the graph",
    "collect coffee mug",
    "generate plan for making coffee",
    "place coffee mug on Tom’s wardrobe"
  ],
  "plan": [
    "go_to(bobs_room1)",
    "pick_up(coffee_mug1)",
    "go_to(kitchen1)",
    "put_inside(coffee_machine1)",
    "turn_on(coffee_machine1)",
    "turn_off(coffee_machine1)",
    "pick_up(coffee_mug1)",
    "go_to(toms_room1)",
    "put_on(wardrobe2)"
  ]
}
```

B APPENDIX – VLM PROMPT STRUCTURE

Describe the image.

```
Return the results in a predefined JSON format as follows:
[
  {
    "name": "object_name",
    "relation": "relation_type",
    "related_to": "related_object_name",
    "states": "object_state",
    "properties": "object_properties"
  }
]
```

Guideline:

```
1. Include only objects that can be moved.
2. Possible states are: open, closed, turned_on, turned_off.
3. Possible relations are: ontop_of, inside_of.

Example Output:
[
  {
    "name": ["bowl",1],
    "relation": "ontop_of",
    "related_to": ["bench", 1],
    "states": "",
    "properties": "black"
  },
  {
    "name": ["apple",1],
    "relation": "inside_of",
    "related_to": ["bowl", 1],
    "states": "",
    "properties": "red"
  },
  {
    "name": ["apple",2],
    "relation": "inside_of",
    "related_to": ["bowl",1],
    "states": "",
    "properties": "green"
  },
  {
    "name": ["bottle",1],
    "relation": null,
    "related_to": null,
    "states": "closed",
    "properties": "green"
  }
]

Do not add objects from list:
<list of assets in the same room and already founded objects>
```

C Appendix – Dataset preparation

Filtered actions from VirtualHome:

Open door, Lock door, Look out window, Movie, Clean, Write school paper, Dust, Play games, Get dressed, Playing video game, Shave, Print out papers, Watch fly, Walk through, Admire art, Gaze out window, Look at painting, Check appearance in mirror, Shut front door, Look at mirror, Write an email, Browse internet, Watch TV, Take shower, Work, Drink, Wash teeth, Wash dishes by hand, Pet cat, Brush teeth, Keep an eye on stove as something is cooking, Open front door, Close door, Pick up phone.

Limited actions for VirtualHome:

”FIND”, ”WALK”, ”GRAB”, ”SWITCHON”, ”TURNTO”, ”PUTBACK”, ”LOOKAT”, ”OPEN”, ”CLOSE”, ”PUTOBJBACK”, ”SWITCHOFF”, ”PUTIN”, ”RUN”,

SayPlan Office tasks: SayPlan Office tasks instructions listed in Table 5.

Table 5: SayPlanOffice Tasks

Tasks description
Close Jason’s cabinet.
Refrigerate the orange left on the kitchen bench.
Take care of the dirty plate in the lunchroom.
Place the printed document on Will’s desk.
Peter is working hard at his desk. Get him a healthy snack.
Hide one of Peter’s valuable belongings.
Wipe the dusty admin shelf.
There is coffee dripping on the floor. Stop it.
Place Will’s drone on his desk.
Move the monitor from Jason’s office to Filipe’s.
My parcel just got delivered! Locate it and place it in the appropriate lab.
Check if the coffee machine is working.
Heat up the chicken kebab.
Something is smelling in the kitchen. Dispose of it.
Heat up the noodles in the fridge, and place it somewhere where I can enjoy it.
Throw the rotting fruit in Dimity’s office in the correct bin.
Safely file away the freshly printed document in Will’s office, then place the undergraduate thesis on his desk.
Make Niko a coffee and place the mug on his desk.
Tobi spilt soda on his desk. Throw away the can and take him something to clean with.
I want to make a sandwich. Place all the ingredients on the lunch table.
Empty the dishwasher. Place all items in their correct locations.
A delegation of project partners is arriving soon. We want to serve them snacks and non-alcoholic drinks. Prepare everything in the largest meeting room. Use items found in the supplies room only.
Serve bottled water to the attendees who are seated in meeting room 1. Each attendee can only receive a single bottle of water.
Locate all 6 complimentary t-shirts given to the PhD students and place them on the shelf in admin.
I’m at the lunch table. Let’s play a prank on Niko. Dimity might have something.