

# Toward Accurate Long-Horizon Robotic Manipulation: Language-to-Action with Foundation Models via Scene Graphs

Sushil Samuel Dinesh and Shinkyu Park

**Abstract**—This paper presents a framework that leverages pre-trained foundation models for robotic manipulation without domain-specific training. The framework integrates off-the-shelf models, combining multimodal perception from foundation models with a general-purpose reasoning model capable of robust task sequencing. Scene graphs, dynamically maintained within the framework, provide spatial awareness and enable consistent reasoning about the environment. The framework is evaluated through a series of tabletop robotic manipulation experiments, and the results highlight its potential for building robotic manipulation systems directly on top of off-the-shelf foundation models.

## I. INTRODUCTION

The primary motivation of this work is to develop a framework that integrates multiple *foundation models*—pre-trained large models—to enable perception, planning, and execution without requiring dedicated end-to-end training or fine-tuning, while maintaining high accuracy. In this framework, users provide verbal commands specifying the desired objective, and the robot perceives the environment, generates a plan, and executes planned tasks directly from this high-level input.

Our framework, as illustrated in Fig. 1, is built on a layered architecture, where each layer is governed by a specialized model and the overall system emerges through their structured interconnection. Many existing approaches, in contrast, employ these models in more constrained ways—for example, as *Large Language Model (LLM)*-based planners without basing their reasoning in spatial understanding [1]–[3], as *Vision-Language Models (VLMs)* that generate trajectories from image overlays but are often error-prone [4]–[6], or as *Vision-Language-Action (VLA) models* that directly map images and language to robot actions but require massive amounts of training data [7]–[9].

In the proposed framework, multiple foundation models are interconnected to address different stages of workflow. In particular, the LLM interprets the user’s request, the VLM provides perception, and a reasoning model generates a detailed task sequence. Each task in the sequence is then executed by a model that integrates a motion planner with a motor controller. Beyond the layered architecture, we incorporate *scene graphs* [10] to provide structured representations of the environment, allowing the models to inform

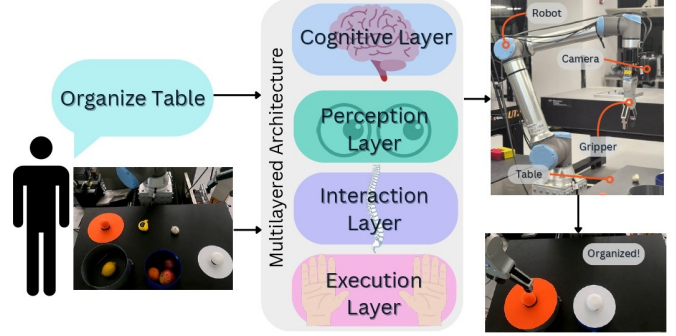


Fig. 1. Overview of the Proposed Framework: The framework is organized into multiple layers, each with distinct capabilities, and is designed to translate high-level natural language commands from the user into an executable sequence of robot actions.

their reasoning more effectively and interpret the workspace with greater accuracy. We evaluate the framework through a series of experiments, ranging from simple object relocation to puzzle solving and long-horizon tasks, demonstrating its versatility and effectiveness.

### A. Comparative Review of Related Work

**LLMs for Planning and Sequencing:** Early efforts to integrate LLMs into robotics generated robot actions by parsing outputs into structured formats like “[Action Name] [Object Name] to [Position Name]”, but were limited to simple tasks in simulation [1]. Subsequent extensions introduced prompt engineering and custom functions, but these approaches still relied on explicitly provided object details and lacked autonomous scene understanding [11].

Subsequent efforts advanced toward multi-model frameworks, for example YOLO-based object recognition with waypoint extraction from human demonstrations [3]. The study of [12] employed exemplars and rule-based systems, improving instruction following but remaining tied to code-based execution. Similarly, [2] demonstrated that LLMs could not introduce new logic (e.g., failure handling) beyond provided examples, motivating direct LLM-driven tool handling (function calling) in a feedback loop.

Collectively, these studies underscore a key limitation: while LLMs excel at symbolic reasoning, they remain weak in integrating reasoning within the physical world.

**VLMs for Perception and Spatial Reasoning:** Several studies have explored the use of overlays—such as arrows or grid-based keypoints—to guide trajectory generation using VLMs [4], [5]. The study of [6] introduced keypoint-based spatial reasoning by deriving 3D coordinates to form constraint functions and sub-goals. While this provided effective

The work was supported by funding from King Abdullah University of Science and Technology (KAUST).

The authors are with the Department of Electrical and Computer Engineering, King Abdullah University of Science and Technology (KAUST), Thuwal, 23955, Saudi Arabia. {sushilsamuel.dinesh, shinkyu.park}@kaust.edu.sa

for tasks such as cloth folding, it struggled to generalize to long-horizon planning, and overall experimental success remained limited.

Other works emphasized dynamic scene understanding. For example, [13] used LLMs to generate object proposals from natural language but remained tied to feature-based representations without explicit spatial modeling. A more complex pipeline explored in [14] incorporated Visual Question Answering (VQA), but still depended on task-specific datasets, bounding box detection, and a fine-tuned InstructBLIP model [15].

In summary, perception in these approaches often remained decoupled from high-level reasoning, leading to brittle pipelines vulnerable to error propagation.

*Scene Graphs and Affordance-based Reasoning:* SayPlan [16] explored scene-graph-based manipulation and planning with affordances for mobile robots, emphasizing semantic search in large multi-room environments. Hydra [17] extended this direction by developing real-time, hierarchical scene graph construction through semantic segmentation, though without considering affordances.

SayCan [18] grounded LLMs in robotics using affordance functions, enabling language-guided task planning with real-world feasibility; however, their approach depended on a large dataset of 68,000 teleoperated actions across 10 robots. VoxPoser [19] derived affordance maps from VLMs to produce 3D representations for trajectory optimization and obstacle avoidance, but it lacked long-horizon planning capabilities.

Overall, while scene graphs and affordances provide strong mechanisms for semantic contextualization, they have not yet been unified with the reasoning depth of foundation models to support robust task sequencing in robots.

*Action Generation and Learning-Based Approaches:* Recent VLA models have significantly advanced robot action generation. PaLM-E [20] introduced short-horizon embodied models, while RT-1 [7] and RT-2 [8] scaled training with large datasets, enabling closed-loop control and language-conditioned reasoning but with limited generalization. GR00T [21] combined a reasoning-capable VLM with a fast diffusion-based motion generator, and Open-X [9] extended this direction through cross-robot training across 22 robots and 160,000 tasks.

Despite these advances, VLA models remain highly data-hungry and struggle to generalize to long-horizon tasks.

**Research Positioning:** Our work addresses these gaps by proposing a structured, layered framework that balances the reasoning strengths of LLMs with the perceptual capabilities of VLMs, organized through scene graphs and executed via conventional motion planning and control. Unlike data-intensive VLA approaches, our framework minimizes the need for task-specific datasets or fine-tuning, thereby reducing the engineering burden of developing dedicated long-horizon robot action models. At the same time, the framework integrates precise object grounding in perception through a state-of-the-art VLM, primary task planning via a powerful LLM, contextual scene understanding through

LLM-VLM dialogue, and persistent spatial reasoning supported by online scene graph updates and task execution by a faster non-reasoning LLM in a feedback loop.

Through experiments demonstrating the successful execution of increasingly complex tasks, this work shows foundation models, when integrated with structured world representations, can effectively bridge the gap between language and action in robotics. The framework positions itself as a middle ground: more generalizable and adaptable than dataset-heavy VLA models, yet more sophisticated and spatially grounded than purely symbolic LLM planners. This dual advantage highlights a promising path toward scalable, adaptable, and semantically informed robotic manipulation.

## B. Paper Organization

Section II presents the framework design, detailing the foundation models assigned to each layer of the architecture; the excerpts of the system prompt commands—that is, the preliminary instructions the LLM is expected to follow—are provided in the Appendix. Section III validates the effectiveness of the framework across a range of manipulation tasks. Finally, Section IV concludes the paper with a summary and directions for future research.

## II. FRAMEWORK DESIGN AND IMPLEMENTATION

### A. Hardware Setup for Target Robotic Systems

Our framework is designed for robotic manipulators equipped with a vision system capable of capturing image data from the workspace. In our experiment setup, as depicted in Fig. 1, we employ a UR10e collaborative robotic arm (6 *DOF*, 1300 *mm* reach, 12.5 *kg* payload) with joint velocity limits for safety, paired with an OnRobot RG6 gripper for handling diverse objects. Perception is supported by a Zivid 2 wrist-mounted 3D camera, which provides high-resolution RGB-D data for reliable scene understanding.

### B. Scene Graph

The scene graph, as depicted in Fig. 2(a), serves as the central knowledge base, encoding spatial relationships, object properties, and semantic details in a format accessible to both foundation models and the motion planner. Implemented with the NetworkX library, it employs a hierarchical JSON representation that balances readability and efficiency. Its design draws inspiration from [16]. An example of scene graph generation and update is provided in the Supplementary Materials.

Each node represents an entity, ranging from the root node denoting the *entire workspace* to individual objects such as an apple or a ball. Nodes capture affordances (e.g., *pickable*), positions, coordinates, and domain knowledge, while edges encode containment relations. Together, they enable the LLM to reason about both physical and semantic constraints during task planning.

Scene graphs can be generated automatically (via GPT-4.1 and Qwen-2.5VL) or manually for complex setups requiring precise ground truth. In our framework, they are dynamically updated by the LLM to ensure consistency as tasks progress:

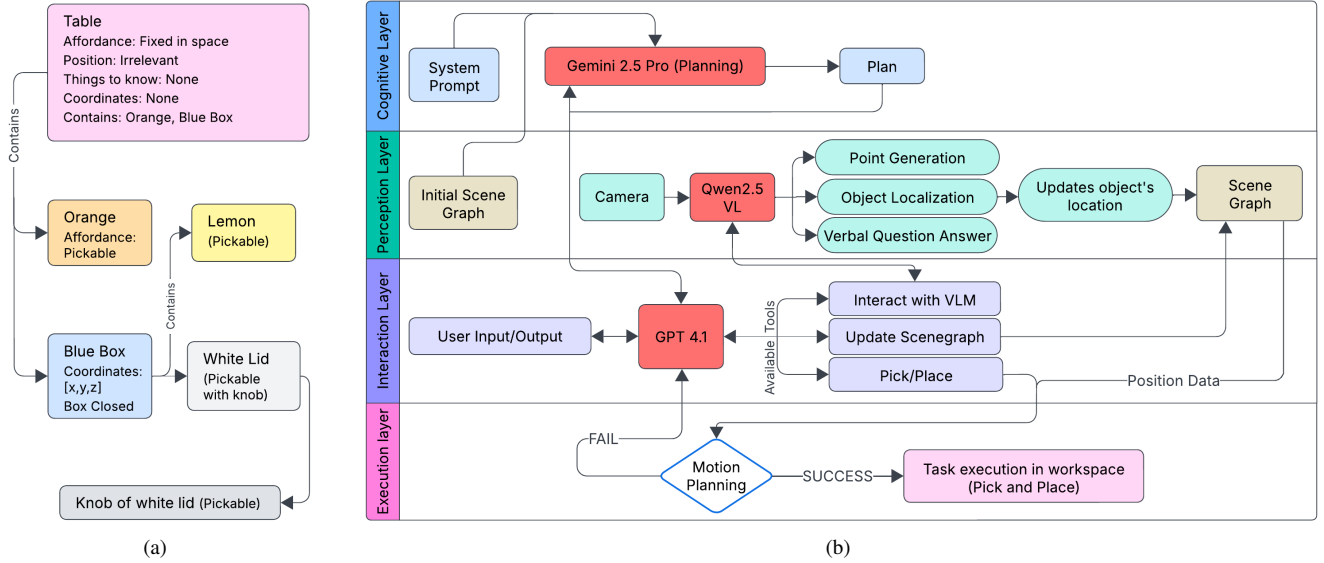


Fig. 2. (a) Scene Graph Structure. (b) System Architecture. The layers are organized in a bottom-up hierarchy. *Execution Layer*: Relies on a conventional motion planner and controller to ensure robust and precise object manipulation. *Interaction Layer*: Utilizes a powerful, non-reasoning model to interpret user instructions and coordinate task execution. *Perception Layer*: Incorporates a VLM with RGB-D input from a 3D camera to provide spatial understanding, object localization, and semantic scene descriptions. *Cognitive Layer*: Employs a reasoning model for advanced long-horizon planning and decision-making.

perception outputs refine object positions, while manipulation actions (e.g., placing an apple in a box) update coordinates and relationships accordingly. Incremental updates are supported through targeted edits, allowing waypoints to be added, properties refined, or user-provided information integrated seamlessly.

### C. Framework Capabilities

Our framework enables robots to interpret natural language instructions and generate long-horizon manipulation plans without task-specific training. Leveraging a 3D camera and VLM, it can perceive and localize novel objects in 3D space, compute precise coordinates in the robot’s frame for grasping and placement, and conduct VQA-style queries between the LLM and VLM to answer semantic questions about the scene (e.g., “do you see some blue object?”). A persistent scene graph maintains object properties and relations, supporting reasoning to infer implicit goals. A reasoning-oriented model produces multi-step, constraint-aware task sequences, while a faster, execution-oriented LLM coordinates motion primitives and continuously updates the scene graph during task execution.

### D. Layered Architecture Design

The proposed framework integrates foundation models with robotic hardware to enable natural language-driven task planning and execution without domain-specific training. Figure 2(b) illustrates the layered architecture of the framework, where each layer is implemented as a modular package within the Robot Operating System 2 (ROS2) framework [22]. The Appendix summarizes the prompts used for GPT-4.1, Qwen2.5-VL, and Gemini 2.5 Pro, the communication between these models, the tools available to GPT-4.1, and the failure-handling procedure used.

In the following, we present detailed explanations of each layer’s functionality and design considerations.

1) *Execution Layer*: The execution layer translates planned tasks into safe and precise robot motions. Nvidia cuRobo’s GPU-accelerated motion planner generates collision-free trajectories that account for robot kinematics, grasp constraints, and workspace boundaries. Coordinated control of the arm and gripper ensures stable pick-and-place operations, while ROS2 integration provides seamless communication with higher-level layers [23].

2) *Interaction Layer*: The interaction layer requests, receives, and executes the high-level task plan produced by the cognitive layer while coordinating with the other layers. Upon receiving the task sequence, it orchestrates the available tools—function calls such as object-manipulation primitives (pick-and-place), perception queries, and scene-graph updates—step by step. These tools are provided to the LLM as callable functions, with each tool call corresponding to the execution of one such function.

At its core, OpenAI’s GPT-4.1—an evolution of GPT-4 [24]—provides natural-language interpretation and function calling, forming a seamless bridge between the user and the robot’s subsystems. During execution, the interaction layer monitors the return outputs of each function call, dynamically re-plans or adjusts the sequence when necessary.

In addition to internal coordination, it communicates directly with the user to provide status updates, request clarifications, or deliver results. This dual role—mediating between layers and interacting with the user—ensures that user intent is reliably realized while maintaining an intuitive, dialogue-driven interface.

3) *Perception Layer*: The perception layer connects visual input with semantic and geometric understanding, providing the spatial context required for robotic manipulation. At its core, it integrates the LLM–VLM dialogue system, where the LLM issues structured queries to the VLM. This interaction enables the robot to “see” and reason about its environment

in a language-driven workflow, enabling perceptual capabilities that traditional vision pipelines would not achieve as fluidly.

Perception tasks follow along two main pathways:

- **Bounding Box Localization:** For accurate object localization, GPT-4.1 queries Qwen2.5-VL to generate bounding boxes around target objects. The process begins with a binary classification query (“Is the target object present?”) to avoid hallucinated detections. For confirmed objects, Qwen2.5-VL outputs pixel-space bounding boxes, which are converted into 3D coordinates using depth data from the robot’s 3D camera and calibrated transformations. Multi-view reconstruction then merges point clouds collected from different camera viewpoints, removes the table plane and outliers, and applies clustering to segment objects. Each cluster is matched to the VLM-derived bounding box point cloud, and the centroid of the matched cluster is returned as the object’s precise 3D location.
- **Specific Point Queries:** For tasks such as finding a free placement spot, the framework employs targeted point queries. Qwen2.5-VL directly outputs pixel coordinates  $(x, y)$  on the 2D image plane, which are projected into 3D space via depth map. This bypasses multi-view reconstruction while still ensuring precise placement coordinates.

Qwen2.5-VL is chosen for its strong spatial reasoning and object-grounding capabilities. Beyond object recognition, it aligns visual features with spatial queries to produce accurate localizations, even in cluttered or ambiguous scenes. Unlike conventional detectors, it requires no retraining on domain-specific datasets, making it highly adaptable. For instance, it can generate bounding boxes or placement points for novel objects and arrangements outside its training distribution, demonstrating robust generalization. This flexibility is particularly valuable in robotic manipulation, where robots often encounter unfamiliar objects and spatial configurations [25].

By combining robust bounding-box detection, flexible point querying, and natural language-driven perception, the perception layer equips the robot with a general-purpose, adaptive understanding of its workspace. This enables reliable spatial awareness for complex manipulation tasks while maintaining real-time performance.

4) *Cognitive Layer:* The cognitive layer is dedicated to reasoning and planning. Google’s Gemini 2.5 Pro Preview [26] generates multi-step task strategies by integrating information from the scene graph, user goals, and tool definitions i.e., the precise descriptions of each tool’s purpose, input parameters, and return values. Since reasoning models are relatively slow, Gemini 2.5 Pro Preview is not suited for the interaction layer. By separating planning (Gemini 2.5 Pro Preview) from execution via the interaction layer (GPT-4.1), the framework combines advanced reasoning with fast, reliable control.

Gemini 2.5 Pro Preview was selected for its strong performance across key dimensions on the LiveBench LLM benchmark [27] at the time of this study. These include

reasoning (essential for spatial and multi-step task planning), language comprehension (crucial for interpreting nuanced natural language instructions), instruction following (necessary for adhering to system prompt rules), and data analysis (particularly relevant for scene graph interpretation).

### III. EXPERIMENTAL EVALUATION

Experiments are designed to evaluate whether our foundation-model-driven framework can generalize across diverse tasks without task-specific training. As noted in [28], there are currently no established benchmarks for assessing the performance of such frameworks. Therefore, rather than pursuing narrow benchmark scores, we demonstrate the proposed framework’s capabilities in perception, reasoning, and planning across a range of settings, including those involving ambiguous or underspecified user requests.

Three classes of experiments are designed to systematically evaluate the framework’s performance and generalizability in complex real-world scenarios, using four key metrics as evaluation criteria:

- *Planning Feasibility (PF):* The feasibility and goal-alignment of the generated task sequences from the cognitive layer.
- *Task Completion Rate (TCR):* The percentage of tasks successfully executed.
- *Scene Graph Handling (SGH):* The correctness of updates made to the world model by the end of each experiment.

To simplify system implementation and the experiment design, we make the following assumptions:

- Since dexterous object manipulation is not the focus of this work, we restrict the scenario to the robot manipulating objects through pick-and-place primitives, with grasping constrained to the centroid of each object.
- API errors and network interruptions are excluded from the failure cases.
- While the motion planner accounts for self-collision and static obstacles (e.g., the table), it does not model collisions with manipulatable objects.
- Our focus is on how the framework accurately updates the scene graph rather than generating it from scratch. Accordingly, initial scene graphs are either provided manually or produced by the LLM at the beginning of each experiment. In particular, the LLM generates the initial graphs for Exps. I-A, I-B, and II-A. For the remaining experiments, the graphs are manually created from scratch (Exp. II-B) or manually created and subsequently modified through simple object or attribute additions and relabeling (Exps. III-A~III-C).

#### A. Experiment I: Testing Fundamental Capabilities

This experiment evaluates whether natural language instructions can be effectively translated into concrete actions requiring recognition, positioning, and contextual reasoning. The following tasks are designed to evaluate the framework’s performance:



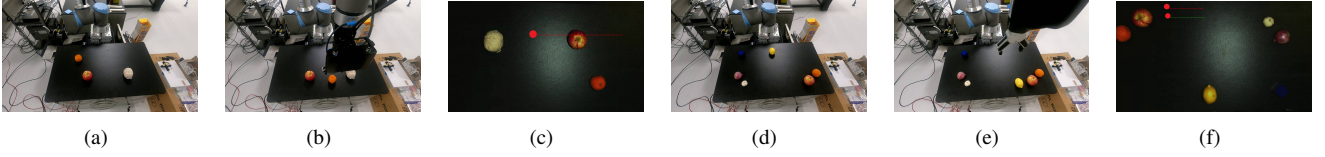


Fig. 3. Experiments I-A and I-B. I-A: (a)–(b) The orange moves from its initial position to between the apple and yarn; (c) shows the VLM-identified point satisfying the “in-between” condition. I-B: (d)–(e) The lemon shifts toward its correct cluster; (f) shows feasible points obtained from the VLM.



Fig. 4. Experiments I-C and I-D. I-C: (a)–(b) The highlighted non-edible object is selected as the odd one out; I-D: (c)–(d) the robot picks only the ingredients required for fried noodles.

- *Experiment I-A (Figs. 3(a)–3(c))*: Relative positioning tasks (e.g., “Move the orange between the apple and yarn”) are designed to assess the stability of the LLM-VLM dialogue and the VLM’s spatial localization capability, particularly its handling of the *in-between* relation. In this setup, the LLM queries the VLM for a point located between the apple and the yarn. The VLM returns the coordinates of the point in the transformed spatial frame, which the LLM uses to update the scene graph and initiate the corresponding pick-and-place operation to move the orange to the designated location.
- *Experiments I-B1 and I-B2 (Figs. 3(d)–3(f))*: Semantic clustering tasks evaluate the framework’s capability for understanding the semantics. In Exp. I-B1, the user request “Move the lone isolated fruit near the other fruits” tests performance under minimal context. In Exp. I-B2, the request “The vegetables and fruits are grouped together respectively. But one fruit is isolated. Move it close to where it belongs” examines how the performance improves as the user provides additional context. In particular, the model’s capability—or lack thereof—can be observed in how well it understands the notions of “lone isolated fruit,” “other fruits,” and the spatial relation “near” in Exp. I-B1, as well as the intended meaning behind “move it closer to where it belongs” in Exp. I-B2.
- *Experiments I-C (Figs. 4(a)–4(b)) and I-D (Figs. 4(c)–4(d))*: Context-based manipulation tasks are conducted to evaluate the framework’s capabilities in outlier detection (e.g., “Transfer the mismatched item from the table to the container”) and recipe-based selection (e.g., “Move the available ingredients for fried noodles into the ingredients box”).

Multiple semantic variations of the user request are employed to evaluate the framework’s contextual understanding, while the initial positions of the manipulated objects are varied in each iteration to create diverse experimental scenarios.

## B. Experiment II: Performance Evaluation in Structured Benchmark-Inspired Scenarios

To evaluate the framework on tasks that require precise robotic reasoning and planning, the following two structured tasks (see Fig. 5) are adopted:

- *Experiment II-A (Figs. 5(a)–5(b))*: Block stacking, designed to test iterative spatial reasoning and scene-graph updating. User requests range from straightforward instructions such as “Stack the blocks with the one in center as base” and “I want you to stack the other blocks on top of the white block” to more abstract directives like “Build something tall using these blocks,” allowing us to observe the model’s ability to interpret user intent—including abstract or open-ended instructions—and adapt its actions accordingly.
- *Experiment II-B (Figs. 5(c)–5(f))*: Tower of Hanoi, a long-horizon puzzle requiring constraint-aware action sequencing. Multiple start-end configurations are tested. To specifically evaluate the cognitive and execution layers, perception was simplified by using AprilTags to identify the base and discs, thereby isolating these layers. Strict adherence to the Tower of Hanoi rules is essential to ensure that each move remains valid and the puzzle’s logical constraints are faithfully respected.

Both tasks are successfully completed across multiple trials, demonstrating the framework’s ability to generate coherent action sequences through language interpretation and scene-graph-based reasoning. This underscores its capability to operate in structured environments that require step-by-step logical planning and reliable scene-graph update.

## C. Experiment III: Advanced Reasoning with Scene Graphs

Further evaluation targets open-ended scenarios requiring multi-step reasoning and semantic categorization (Exps. III-A~III-C) as well as occlusion handling (Exp. III-C). In both Exps. III-B and III-C, the framework must additionally leverage the VLM’s generalization capability to localize the lid knobs for grasping, as illustrated in Fig. 6.

The tasks include:

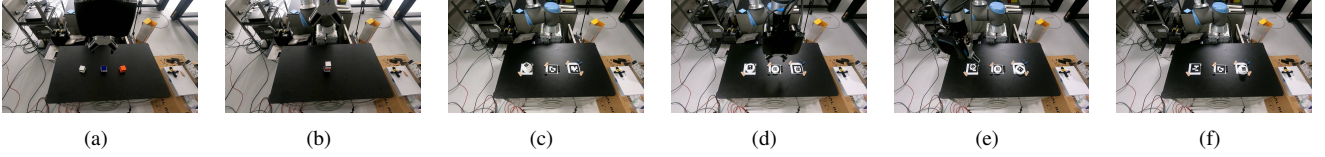


Fig. 5. Experiments II-A and II-B. II-A: (a)–(b) The blocks progress from their initial arrangement on the table to a fully stacked structure. II-B: (c)–(f) The robot solves the Tower of Hanoi puzzle, moving discs step by step from one base to another adhering to the rules of the game.

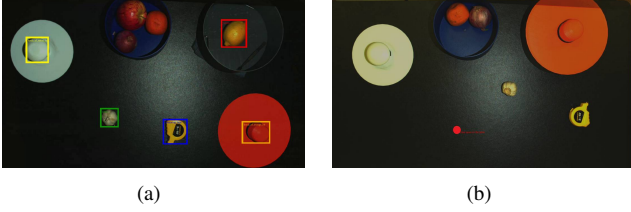


Fig. 6. (a) Yellow bounding boxes show the VLM’s ability to localize fine-grained affordances, such as lid knobs. (b) The bright red point illustrates the VLM’s spatial awareness in assigning a temporary placement location for the lid.

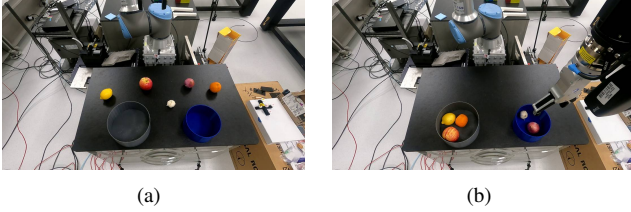


Fig. 7. Experiment III-A. (a)–(b) Items are sorted into two boxes according to inferred categories (fruits vs. vegetables), with fruits placed in the larger container and vegetables in the smaller one.

- *Experiment III-A (Figs. 7(a)–7(b))*: Autonomous sorting, where items are grouped into inferred categories (e.g., fruits vs. vegetables). Two containers are provided: a large box and a small box. Because the table contains more fruits than vegetables, the proposed framework is expected to (1) correctly distinguish fruits from vegetables, (2) allocate fruits to the large box, and (3) allocate vegetables to the small box. This setup evaluates both categorization accuracy and container assignment based on relative quantities—all from the simple user input of “Put the objects into boxes in an organized manner.”
- *Experiment III-B (Figs. 8(a)–8(d))*: In the autonomous table-organization task, the scene graph includes the grey box and the blue box (positioned on the left and right of the table, respectively), labeled as a toolbox and a food-items box—objects and labels added manually. At the start of the experiment, a lemon is intentionally misplaced in the toolbox (Fig. 8(a)). The framework must detect this inconsistency and relocate the lemon to the food-items box (Fig. 8(b)) before proceeding to place all remaining objects into their designated boxes (Fig. 8(c)) and closing each with its corresponding lid (Fig. 8(d)). With only the user command “Organize the table”, the framework is required to infer the full task sequence by extracting relevant details from the initial scene graph, generating and executing a plan, and updating the scene graph online throughout execution.

TABLE I  
EXPERIMENTAL RESULTS (*PF*: PLANNING FEASIBILITY, *TCR*: TASK COMPLETION RATE, *SGH*: SCENE GRAPH HANDLING.)

Exp.	<i>PF</i> (%)	<i>TCR</i> (%)	<i>SGH</i> (%)	Exp.	<i>PF</i> (%)	<i>TCR</i> (%)	<i>SGH</i> (%)
I-A	100	100	100	II-A	100	100	100
I-B1	100	20	100	II-B	100	100	100
I-B2	100	100	100	III-A	100	100	100
I-C	100	100	100	III-B	95	75	100
I-D	100	80	100	III-C	80	60	100

- *Experiment III-C (Figs. 8(e)–8(f))*: This experiment extends Exp. III-B by introducing an occlusion scenario in the autonomous table-organization task. Initially, the toolbox is closed (Fig. 8(e)). The framework must interpret this context via the scene graph, and then leverage the VLM to identify a temporary location on the table (Fig. 6(b)) for placing the toolbox lid (Fig. 8(f)). The scene graph is then updated to record the lid’s temporary position, allowing the framework to effectively handle the occlusion. The subsequent tasks mirror Exp. III-B: relocating objects to their designated boxes, organizing the table, and closing the boxes with their lids. As before, the only user input is the high-level instruction “Organize the table”, with the system autonomously inferring and executing all steps while continuously updating the scene graph during execution.

#### D. Analysis and Discussions

We conducted 10 trials for Exp. I-A, 20 for Exp. III-B, and 5 trials each for all remaining experiments. As summarized in Table I, the framework consistently achieves high planning feasibility (*PF*) and perfect scene-graph handling (*SGH*). Specifically,  $PF \geq 95\%$  in all experiments except Exp. III-C (80%), highlighting the difficulty of addressing occlusion scenarios, while *SGH* remains at 100% across all cases. Task completion rate (*TCR*) reflects both the clarity of user instructions and the complexity of scene understanding: fundamental and structured tasks such as Exps. I-A, I-B2, I-C, II-A, II-B, and III-A reach  $TCR = 100\%$ , whereas Exp. I-D shows a moderate decrease to 80%. In contrast, the underspecified instructions in Exp. I-B1 limited the VLM’s performance, as the LLM could not provide sufficient context for spatial reasoning, resulting in a *TCR* of only 20% despite  $PF = 100\%$ . Rephrasing the user input with richer context in Exp. I-B2 restored higher *TCR*.

The most challenging scenarios arise in cluttered or occluded scenes in Exp. III, where Exp. III-B achieves *TCR* of 75% and Exp. III-C drops further to 60%. Long-horizon tasks such as the Tower of Hanoi (Exp. II-B) and occlusion handling (Exp. III-C) further highlight the benefit of decoupling extensive planning—managed by a reasoning

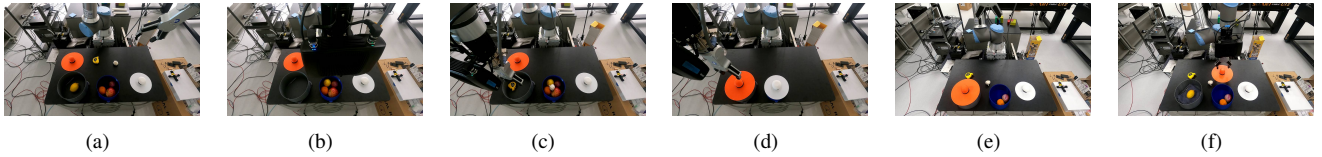


Fig. 8. Experiments III-B and III-C. III-B: (a)–(b) The misplaced lemon is transferred from the toolbox to the food items box; (c) the remaining table objects are organized, (d) and each box is closed with its appropriate lid. III-C: (e)–(f) The robot interprets occlusion by opening the initially closed toolbox, placing the lid at a VLM-identified temporary point, and then proceeding as in III-B.

LLM—from execution—handled by a non-reasoning LLM. Overall, across fundamental capabilities (Exps. I-A~I-D), structured benchmarks (Exps. II-A and II-B), and advanced reasoning (Exps. III-A~III-C), the framework reliably maintains robust world-model representations, with failures concentrated in cases of ambiguous language and increased physical complexity. As a result, it achieves high *TCR* in most experiments. The few VLM errors—limited to Exp. I-B1, where the LLM could not supply adequate contextual cues, and Exp. III-C, where the scene was too visually cluttered to parse—illustrate these two sources of difficulty.

**Strengths:** Leveraging scene graphs enabled multi-step planning through context retrieval from the graphs, allowing the framework to better interpret high-level user requests (e.g., “Organize the table” in Exps. III-B and III-C). The VLM demonstrated strong generalization to novel objects and affordances (e.g., lid-knob localization and temporary placement points) without retraining. Furthermore, decoupling reasoning from execution produced coherent plans while ensuring reliable, deterministic low-level control, with execution calls handled by a non-reasoning model in a feedback loop.

**Limitations:** The framework remains sensitive to ambiguous language: sparse user requests occasionally caused the LLM to issue queries to the VLM without sufficient context, leading to errors (as observed in Exp. I-B1). This illustrates error propagation between layers, particularly when performing spatial localization through the VLM. At the execution level, failures also occasionally arose from collisions with manipulatable objects, which were not modeled as dynamic obstacles, thereby reducing *TCR* in cluttered scenes (Exp. III-B).

**Takeaway:** The layered design of the framework offers a practical middle ground—more adaptable than dataset-heavy VLA systems and more robust than LLM-only and VLM-only planners, particularly in long-horizon, semantically constrained manipulation.

#### IV. CONCLUSION

This study introduced a novel foundation-model framework that integrates multiple models across different layers with a scene graph representation of the environment, enabling natural-language manipulation without task-specific training. Experiments demonstrated near-ceiling performance in planning and scene-graph handling, strong perceptual generalization, and reliable execution in both structured and open-ended tasks. Remaining bottlenecks include handling linguistic ambiguity and mitigating collisions arising from the lack of dynamic obstacle modeling at the execution level.

As future research, we aim to extend the framework with dexterous object manipulation and to further explore the use of VLMs for manipulating deformable objects and those with complex geometries.

#### APPENDIX

Here, we provide only excerpts of the prompts used in our framework, with “...” indicating omitted details. The complete prompt specifications are available in Supplementary Materials.

- **GPT-4.1:** “You are a 6-DoF UR10e arm with a two-finger gripper ... Stop if you encounter any failures and let the user know.” Another variation of this prompt is used for Exp. II-B to use AprilTags instead of VLMs.
- To prevent undesired behaviors, rules are embedded in the system prompt and tool descriptions. For example, the system prompt includes the rule “When robot movement is involved (pick or place), you may execute only one movement based tool call at a time.” so that GPT can keep track of feedback from the motion planner. Another example from a tool’s description is “do not call this function after pick and before place”, which is necessary because an object in the end effector would obscure the camera. When GPT forwards a request to Gemini, these rules are passed along with the available tool definitions.
- **Gemini 2.5 Pro:** “You are a robotic arm. Provide the correct action sequence ... [Available Tools] [Initial Scene Graph].” Gemini also gets rules for generating the exhaustive plan, for example: “When using `scan_and_update_coordinates_in_scene_graph`, scan as many `VISIBLE` objects at once...”
- **Qwen2.5 VL:** To Retrieve Bounding Box: (1) “Do you see [object]? Answer 1 or 0.” (2) “Output coordinates of all objects in JSON.” To Retrieve Specific Point: “Point to [request from GPT] and output a single coordinate in XML.”

#### Model Communication:

- **GPT-4.1 ↔ Gemini 2.5 Pro:** GPT forwards the user request—along with the scene graph and available tools—to Gemini and receives an exhaustive plan for task execution.
- **GPT-4.1 ↔ Qwen 2.5 VL:** GPT prompts Qwen for coordinates or object names to localize and update scene-



graph positions; Qwen also answers VQA queries. No chatlog is stored on Qwen.

- GPT-4.1  $\leftrightarrow$  Motion Primitives / Scene Graph: GPT passes parameters to motion or scene-graph functions and receives success/failure feedback.
- A parameter is an input a tool uses to operate—like `object_name` in `pick(object_name)`. The function returns text for VQA with VLM, or a success/failure message (with a reason if it fails) for tasks like object localization, scene graph editing, or motion planning.

#### Available Tools for GPT 4.1:

- Motion Primitives: `pick object` and `place object`, implemented with Nvidia cuRobo
- Perception: ask VQA VLM, scan and update coordinates in scene graph (Given a set of object names, detect visible objects, localize their 3D poses using VLM issued bounding boxes, and update the scene graph accordingly.), get a specific coordinate point using VLM (GPT provides a prompt and VLM returns the coordinate of the points satisfying that prompt. e.g. point in-between object 1 and object 2). Points are typically used in our implementation to get locations to place objects. However, in Exp. II-A, points were used for both picking and placing since the scan and update tool was not provided to LLMs. For Exp. II-B we use get current position of visible AprilTags instead of VLM based functions.
- Scene Graph: The scene graph can be modified using two key tools: `add object to scene graph (object name, attributes)` to insert a new object with the specified attributes, and `edit scene graph(object name, attribute name, value)` to update an existing object’s attributes, such as containment relationships. For example, if the robot removes an object from a box and the box becomes empty, GPT would update the scene graph with `edit_scene_graph("box", "contains", None)`.
- Cognitive Layer: Task sequence planning using a reasoning model (Gemini 2.5 Pro).

**Failure Handling:** GPT tracks every function call and feedback. If a step fails, it alerts the user, re-plans, and retries. After repeated failures, it may suggest skipping the object or ask the user to reposition it when it’s not visible—something we experienced and followed in Exps. III-B and III-C.

#### REFERENCES

- [1] H. You, Y. Ye, T. Zhou, Q. Zhu, and J. Du, “Robot-enabled construction assembly with automated sequence planning based on chatgpt: Robogpt,” *Buildings*, vol. 13, no. 7, 2023.
- [2] S. Huang, Z. Jiang, H. Dong, Y. Qiao, P. Gao, and H. Li, “Instruct2act: Mapping multi-modality instructions to robotic actions with large language model,” *arXiv:2305.11176*, 2023.
- [3] N. Wake, A. Kanehira, K. Sasabuchi, J. Takamatsu, and K. Ikeuchi, “Gpt-4v(ision) for robotics: Multimodal task planning from human demonstration,” *IEEE Robotics and Automation Letters*, vol. 9, no. 11, pp. 10 567–10 574, 2024.
- [4] S. Nasiriany *et al.*, “Pivot: iterative visual prompting elicits actionable knowledge for vlms,” in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML’24. JMLR.org, 2024.
- [5] K. Fang, F. Liu, P. Abbeel, and S. Levine, “Moka: Open-world robotic manipulation through mark-based visual prompting,” *Robotics: Science and Systems (RSS)*, 2024.
- [6] W. Huang, C. Wang, Y. Li, R. Zhang, and L. Fei-Fei, “Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation,” *arXiv:2409.01652*, 2024.
- [7] A. Brohan *et al.*, “RT-1: Robotics Transformer for Real-World Control at Scale,” in *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023.
- [8] B. Zitkovich *et al.*, “Rt-2: Vision-language-action models transfer web knowledge to robotic control,” in *Proceedings of The 7th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 229. PMLR, 06–09 Nov 2023, pp. 2165–2183.
- [9] A. O’Neill *et al.*, “Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 6892–6903.
- [10] X. Chang, P. Ren, P. Xu, Z. Li, X. Chen, and A. Hauptmann, “A comprehensive survey of scene graphs: Generation and application,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 1, pp. 1–26, 2023.
- [11] N. Wake, A. Kanehira, K. Sasabuchi, J. Takamatsu, and K. Ikeuchi, “Chatgpt empowered long-step robot control in various environments: A case application,” *IEEE Access*, vol. 11, pp. 95 060–95 078, 2023.
- [12] M. Xu *et al.*, “Creative robot tool use with large language models,” *arXiv:2310.13065*, 2023.
- [13] B. Chen *et al.*, “Open-vocabulary queryable scene representations for real world planning,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 11 509–11 522.
- [14] J. Gao *et al.*, “Physically grounded vision-language models for robotic manipulation,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 12 462–12 469.
- [15] W. Dai *et al.*, “Instructblip: towards general-purpose vision-language models with instruction tuning,” in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS ’23. Red Hook, NY, USA: Curran Associates Inc., 2023.
- [16] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Sunderhauf, “Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning,” in *Proceedings of The 7th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 229. PMLR, 06–09 Nov 2023, pp. 23–72.
- [17] N. Hughes, Y. Chang, and L. Carlone, “Hydra: A real-time spatial perception system for 3D scene graph construction and optimization,” in *Robotics: Science and Systems XVIII*, 2022.
- [18] B. Ichter *et al.*, “Do as I can, not as I say: Grounding language in robotic affordances,” in *Proceedings of The 6th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 205. PMLR, 14–18 Dec 2023, pp. 287–318.
- [19] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, “Voxposer: Composable 3d value maps for robotic manipulation with language models,” in *Proceedings of The 7th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 229. PMLR, 06–09 Nov 2023, pp. 540–562.
- [20] D. Driess *et al.*, “Palm-e: an embodied multimodal language model,” in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML’23. JMLR.org, 2023.
- [21] NVIDIA *et al.*, “Gr00t n1: An open foundation model for generalist humanoid robots,” *arXiv:2503.14734*, 2025.
- [22] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, “Robot operating system 2: Design, architecture, and uses in the wild,” *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.
- [23] B. Sundaralingam *et al.*, “Curobo: Parallelized collision-free robot motion generation,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 8112–8119.
- [24] OpenAI *et al.*, “GPT-4 technical report,” *arXiv:2303.08774*, 2024.
- [25] S. Bai *et al.*, “Qwen2.5-vl technical report,” *arXiv:2502.13923*, 2025.
- [26] G. Comanici *et al.*, “Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities,” *arXiv:2507.06261*, 2025.
- [27] C. White *et al.*, “Livebench: A challenging, contamination-free LLM benchmark,” in *The Thirteenth International Conference on Learning Representations*, 2025.
- [28] Y. Hu *et al.*, “Toward general-purpose robots via foundation models: A survey and meta-analysis,” *arXiv:2312.08782*, 2024.



# Supplementary Materials

## I. PROMPT USED FOR GPT 4.1

### A. GPT 4.1

The following system prompt is used across all experiments for GPT-4.1.

“You are a 6DoF casual and friendly Universal Robotics UR10e arm with a two finger gripper. Your task is to initiate the conversation and get a request from the user first. Then, pass it to the `plan_using_advanced_llm` which can give you the right order of tool calls with parameters. Don’t ask questions to the user and simply pass his request to the advanced LLMs since they also have access to the scene graph and available tools. Based on the output of `plan_using_advanced_llm` planner you MUST STRICTLY ADHERE to its plan and execute its plan. When robot movement is involved (pick or place), you may execute only one movement based tool call at a time. Pick, modifying the scene graph can be called at once but you cannot call pick and place at once. Observe the feedback before going to the next manipulation/movement tool call. Once you receive the plan, ask user for confirmation. Once he confirms you can execute until everything is done. Stop if you encounter any failures and let user know.”

### B. Tool Descriptions

In addition to the aforementioned system prompt for GPT 4.1, the tool descriptions are also added to the system prompt to enable tool use (function calling) with GPT 4.1.

For object manipulation, the following tools were used.

- **pick\_object:** “Makes the robot pick a provided object. The name MUST precisely match what is in the database/scene graph whose coordinate is available.”  
**Parameter:** `object_name`
- **place\_object:** “Makes the robot to place an object in hand safely at the provided place name. The name MUST precisely match what is in the database/scene graph whose coordinate is available.”  
**Parameter:** `place_position_name`

For VLM interaction, the following tools were used.

- **ask\_vqa\_vlm:** “This is a VLM (QwenVL 2.5). You can ask VQA. The VLM will answer with whatever you want to know. Only for single Q&A and not conversations since the chat history is not stored.”  
**Parameter:** `query_to_vlm`
- **scan\_and\_update\_coordinates\_in\_scene\_graph:** “DO NOT CALL AFTER PICK\_OBJECT. The camera mounted on the robotic arm will scan and update the position of the requested object using a VLM. The camera mounted near gripper needs unobstructed view when scanning and it cannot see hidden objects. Therefore, do not call this function after pick and before place. This function fills/updates the scene graph nodes (targets) with coordinates by itself. This can only scan identifiable individual

objects. To simply get points you need to user `get_a_specific_coordinate_point_using_vlm.`”

**Parameter:** `targets_to_scan`

- **get\_a\_specific\_coordinate\_point\_using\_vlm:** “DO NOT CALL AFTER PICK\_OBJECT. The camera mounted on the robotic arm will look at the workspace. Then VLM will give you the specific point you ask for in the workspace. Output given to you will be of the form `[x,y,z]`. Based on response you need to add/update the scene graph with received output coordinates by yourself. The gripper should also be free of any objects when scanning. Therefore DO NOT call this function after `pick_object` and before `place_object`. Strictly do NOT add ‘I want `[x,y,z]` coordinate’ in prompt and keep the prompt SHORT. The camera ALWAYS takes the TOP VIEW Photo of the workspace/table.”

**Parameter:** `prompt_to_vlm`

For the Tower of Hanoi experiment (Exp. II-B), VLM was not used and the following tool was used instead to localize Apriltags.

- **get\_current\_position\_of\_visible\_apriltags:** “This function lets you know of the TAG ID and Position `[x,y,z]` of all the currently visible Apriltags. But ONLY VISIBLE Objects with Apriltags are captured. Obscured objects are not visible. Note that camera captures top down view of table.”  
**Parameter:** `trigger`

For modifying the scene graph, the following tools were used.

- **add\_object\_to\_scenegraph:** “Adds a new object in the scene graph with specified parameters.”  
**Parameters:** `object_name, affordance, position_in_cartesian_space, things_to_know, coordinates, contains`
- **edit\_scenegraph:** “Edit the attribute of any node that is already present in the scene graph.”  
**Parameters:** `node_name, attribute_name, value`

The following tool was used to send the required data to Gemini.

- **plan\_using\_advanced\_llm:** “You will get a detailed plan from advanced LLM to execute. This ensures high success rate.”  
**Parameter:** `request_from_user`

## II. PROMPTS USED FOR GEMINI 2.5 PRO

### A. Default Prompt

This is the default prompt used in all the experiments except Exp. II-B.

“You are a robotic arm. Your task is to give the right sequence to achieve the user request `[request_from_user]`

Important note: 1) If you move (pick or place) an object, you need to update its position (coordinates) again before attempting another pick or place. If you don't do that, the robot will unintentionally approach the past position available in the scene graph. 2) Scanning is time-consuming. So update the position of manipulated objects if and only if you want to manipulate it again which requires the latest position. 3) Update the scene graph as required. But don't waste time in scanning newly updated positions unless you plan to manipulate those objects further. 4) **MANDATORY:** You are **PROHIBITED** to use `get_a_specific_coordinate_point_using_vlm` AFTER `pick_object` since an object held in hand will block cameras completely. **STRICT RULE. MUST FOLLOW!!** 5) Make sure to mark any placeholder values in case it depends on a previous function call in order for the actual action executing LLM to understand properly. 6) You are far more intelligent (way larger model) than the ones used by VLM. So only use VLM as your eyes and not for anything that involves logic, reasoning and wider knowledge base. Use the VQA and Monologue to perceive—that's it. 7) When using `scan_and_update_coordinates_in_scene_graph`, scan as many **VISIBLE** objects at once since scanning one by one can take some time since the robot needs to reach several vantage points to construct pointcloud for processing.

You can use the following functions: [Available Tools]

The following is the scene graph representation available currently. [Initial Scene Graph]"

The Available Tools are identical to the tool descriptions provided to GPT 4.1 in Section I-B of Supplementary Materials. The Initial Scene Graph is the available scene graph in JSON sent to the LLM in text format.

### B. Alternative Prompt

This alternative prompt is used only for the Tower of Hanoi experiment (Exp. II-B) involving Apriltags.

"You are a robotic arm. Your task is to give the right sequence to achieve the user request [request\_from\_user]"

Important note: 1) If you move (pick or place) an object, you need to update its position (coordinates) again before attempting another pick or place. If you don't do that, the robot will unintentionally approach the past position available in the scene graph. 2) **NEVER EVER** use `get_current_position_of_visible_apriltags` between pick-and-place. Because an object in the end effector will block the view of the workspace. So you can use this function only after placing whatever is in hand. 3) Use the `get_current_position_of_visible_apriltags` function to get the latest position. Make sure to update in the scene graph after fetching the values. So that pick-and-place can use that. 4) Make sure to mark any placeholder values in case it depends on a previous function call in order for

the actual action executing LLM to understand properly. 5) Apriltags are used for localization. But Apriltags are only seen by the camera if it is not obscured when capturing the top-down view of the table/workspace. So don't attempt to see potentially obscured objects. 6) When using `place_object`, use the name of the object that will be underneath the current object, rather than using generic labels like `base_1`, `base_2`, `base_3` unless you are specifically placing on the base surface.

You can use the following functions: [Available Tools]

The following is the scene graph representation available currently. [Initial Scene Graph]"

## III. PROMPTS USED FOR QWEN2.5 VL

### A. Getting Bounding Box

The first prompt verifies the presence of a given object to prevent the VLM from hallucinating non-existent ones. This step ensures that the VLM explicitly confirms whether the object is present. Only the objects confirmed to exist are subsequently sent to the VLM for bounding box generation.

- Prompt: "Do you see [object name] in the image. Answer strictly in binary. 1 or 0."
- System Prompt: "Output is STRICTLY Binary. 1 or 0."

Once the objects present in the frame are confirmed, their bounding boxes can be obtained using the following prompt.

- Prompt: "Outline the position of object names and output all the coordinates in the JSON format."
- System Prompt: Strictly maintain format

```
1 [{"bbox_2d": [integer, integer,
2      integer, integer], "label": "
   obj_name"}],
3 ...
4 ]
```

In this case, the system prompt specifies the desired output JSON format required for parsing.

### B. Getting Specific Point

The system prompt defines the required output format for parsing. The prompt used is shown below.

- System Prompt: "STRICTLY ADHERE TO OUTPUT FORMAT <points x y>object</points>. Single Coordinate. STRICTLY ADHERE TO THIS FORMAT!!!!"
- Prompt: "Point to the prompt from GPT. STRICTLY output a SINGULAR coordinate in XML format <points x y>object</points>"

## IV. MODEL VARIANTS AND ACCESS

- GPT 4.1 - gpt-4.1-2025-04-14. Accessed via the OpenAI API.
- Gemini 2.5 Pro - gemini-2.5-pro-preview-05-06. Accessed via the Google Gemini API.

- Qwen2.5-VL - 32B and 72B. Accessed via OpenRouter, with the specific variant selected based on availability.

## V. SCENE GRAPH EXAMPLE

The following is the initial scene graph provided to the LLMs for Exp. III-A.

```

1 {
2   "workspace": {
3     "affordance": [
4       "None"
5     ],
6     "contains": [
7       "table"
8     ],
9     "position_in_cartesian_space": "
10      irrelevant",
11     "things_to_know": "None",
12     "coordinates": []
13   },
14   "table": {
15     "affordance": [
16       "fixed in space"
17     ],
18     "contains": [
19       "small_box",
20       "large_box",
21       "orange",
22       "apple",
23       "lemon",
24       "garlic",
25       "red_onion"
26     ],
27     "position_in_cartesian_space": "
28      irrelevant. coordinates not
29      available as table refers to the
30      whole accessible workspace. You
31      need specific point in the table if
32      you want to place something on the
33      table.",
34     "things_to_know": "None",
35     "coordinates": []
36   },
37   "orange": {
38     "affordance": [
39       "pickable",
40       "edible"
41     ],
42     "contains": [],
43     "position_in_cartesian_space": "
44      centroid_can_be_obtained",
45     "things_to_know": "A small, round,
46      orange-colored fruit.",
47     "coordinates": []
48   },
49   "apple": {
50     "affordance": [
51       "pickable",
52       "edible"
53     ],
54     "contains": [],
55     "position_in_cartesian_space": "
56      centroid_can_be_obtained",
57     "things_to_know": "A medium-sized,
58      round fruit with red and yellow
59      striped skin.",
60     "coordinates": []
61   },
62   "lemon": {
63     "affordance": [
64       "pickable",
65       "edible"
66     ],
67     "contains": [],
68     "position_in_cartesian_space": "
69      centroid_can_be_obtained",
70     "things_to_know": "A small, oval,
71      yellow-colored fruit.",
72     "coordinates": []
73   },
74   "garlic": {
75     "affordance": [
76       "pickable",
77       "edible"
78     ],
79     "contains": [],
80     "position_in_cartesian_space": "
81      centroid_can_be_obtained",
82     "things_to_know": "A small, bulbous,
83      off-white vegetable with a papery
84      outer skin.",
85     "coordinates": []
86   },
87   "red_onion": {
88     "affordance": [
89       "pickable",
90       "edible"
91     ],
92     "contains": [],
93     "position_in_cartesian_space": "
94      centroid_can_be_obtained",
95     "things_to_know": "A bulb-shaped
96      vegetable with a deep purple
97      outer layer.",
98     "coordinates": []
99   },
100  "small_box": {
101    "affordance": [
102      "pickable"
103    ],
104    "contains": [],
105    "position_in_cartesian_space": "
106      Position is explicitly defined",
107    "things_to_know": "This is fixed in
108      table. This is a cylindrical box.
109      It has a smaller radius.",
110    "coordinates": [0.19957663118839264,
111      -0.6754058599472046,
112      0.14970232427120209]
113  },
114  "large_box": {
115    "affordance": [
116      "pickable"
117    ],
118    "contains": [],
119    "position_in_cartesian_space": "
120      Position is explicitly defined.
121      This is a cylindrical box. It has
122      a larger radius.",
123    "things_to_know": "This is fixed in
124      table",
125    "coordinates": [-0.17225371301174164,
126      -0.6708526611328125,
127      0.14970232427120209]
128  }
129 }
```

```

48   "coordinates": []
49 },
50 "lemon": {
51   "affordance": [
52     "pickable",
53     "edible"
54   ],
55   "contains": [],
56   "position_in_cartesian_space": "
57     centroid_can_be_obtained",
58   "things_to_know": "A small, oval,
59     yellow-colored fruit.",
60   "coordinates": []
61 },
62 "garlic": {
63   "affordance": [
64     "pickable",
65     "edible"
66   ],
67   "contains": [],
68   "position_in_cartesian_space": "
69     centroid_can_be_obtained",
70   "things_to_know": "A small, bulbous,
71     off-white vegetable with a papery
72     outer skin.",
73   "coordinates": []
74 },
75 "red_onion": {
76   "affordance": [
77     "pickable",
78     "edible"
79   ],
80   "contains": [],
81   "position_in_cartesian_space": "
82     centroid_can_be_obtained",
83   "things_to_know": "A bulb-shaped
84     vegetable with a deep purple
85     outer layer.",
86   "coordinates": []
87 },
88 "small_box": {
89   "affordance": [
90     "pickable"
91   ],
92   "contains": [],
93   "position_in_cartesian_space": "
94     Position is explicitly defined",
95   "things_to_know": "This is fixed in
96     table. This is a cylindrical box.
97     It has a smaller radius.",
98   "coordinates": [0.19957663118839264,
99     -0.6754058599472046,
100     0.14970232427120209]
101 },
102 "large_box": {
103   "affordance": [
104     "pickable"
105   ],
106   "contains": [],
107   "position_in_cartesian_space": "
108     Position is explicitly defined.
109     This is a cylindrical box. It has
110     a larger radius.",
111   "things_to_know": "This is fixed in
112     table",
113   "coordinates": [-0.17225371301174164,
114     -0.6708526611328125,
115     0.14970232427120209]
116 }
```

```

98     }
99 }
0.14970232427120209]

```

By the end of Exp. III-A, the LLM had made several modifications to the scene graph, as shown below.

```

1  ras
2  {
3    "workspace": {
4      "affordance": [
5        "None"
6      ],
7      "contains": [
8        "table"
9      ],
10     "position_in_cartesian_space": "
        irrelevant",
11     "things_to_know": "None",
12     "coordinates": []
13   },
14   "table": {
15     "affordance": [
16       "fixed in space"
17     ],
18     "contains": [
19       "small_box",
20       "large_box"
21     ],
22     "position_in_cartesian_space": "
        irrelevant. coordinates not available
        as table refers to the whole
        accessible workspace. You need
        specific point in the table if you
        want to place something on the table
        .",
23     "things_to_know": "None",
24     "coordinates": []
25   },
26   "orange": {
27     "affordance": [
28       "pickable",
29       "edible"
30     ],
31     "contains": [],
32     "position_in_cartesian_space": "inside
        large_box",
33     "things_to_know": "A small, round, orange
        -colored fruit.",
34     "coordinates": []
35   },
36   "apple": {
37     "affordance": [
38       "pickable",
39       "edible"
40     ],
41     "contains": [],
42     "position_in_cartesian_space": "inside
        large_box",
43     "things_to_know": "A medium-sized, round
        fruit with red and yellow striped
        skin.",
44     "coordinates": []
45   },
46   "lemon": {
47     "affordance": [
48       "pickable",
49       "edible"

```

```

50   ],
51   "contains": [],
52   "position_in_cartesian_space": "inside
        large_box",
53   "things_to_know": "A small, oval, yellow-
        colored fruit.",
54   "coordinates": []
55 },
56 "garlic": {
57   "affordance": [
58     "pickable",
59     "edible"
60   ],
61   "contains": [],
62   "position_in_cartesian_space": "inside
        small_box",
63   "things_to_know": "A small, bulbous, off-
        white vegetable with a papery outer
        skin.",
64   "coordinates": []
65 },
66 "red_onion": {
67   "affordance": [
68     "pickable",
69     "edible"
70   ],
71   "contains": [],
72   "position_in_cartesian_space": "inside
        small_box",
73   "things_to_know": "A bulb-shaped
        vegetable with a deep purple outer
        layer.",
74   "coordinates": []
75 },
76 "small_box": {
77   "affordance": [
78     "pickable"
79   ],
80   "contains": [
81     "garlic",
82     "red_onion"
83   ],
84   "position_in_cartesian_space": "Position
        is explicitly defined",
85   "things_to_know": "This is fixed in table
        . This is a cylindrical box. It has a
        smaller radius.",
86   "coordinates": [
87     0.19957663118839264,
88     -0.6754058599472046,
89     0.1497023242712021
90   ]
91 },
92 "large_box": {
93   "affordance": [
94     "pickable"
95   ],
96   "contains": [
97     "orange",
98     "apple",
99     "lemon"
100  ],
101   "position_in_cartesian_space": "Position
        is explicitly defined. This is a
        cylindrical box. It has a larger
        radius.",
102   "things_to_know": "This is fixed in table
        ",

```



```
103     "coordinates": [  
104         -0.17225371301174164,  
105         -0.6708526611328125,  
106         0.1497023242712021  
107     ]  
108 }  
109 }
```

---

The scene graph handling in this case is considered successful, as the hierarchies were updated: the fruits and vegetables that were originally on the table were moved into two boxes. Specifically, `garlic` and `red_onion` were placed in the `small_box`, while `orange`, `apple`, and `lemon` were placed in the `large_box`.