

Memory–Environment Dual-Grounding for Embodied Agents: Active Information Seeking Beyond Passive Observation

Jae-Woo Choi^{1,2*}, Sehoon Oh^{3*}, Ue-Hwan Kim^{3†}

¹Electronics and Telecommunications Research Institute (ETRI)

²University of Science and Technology (UST)

³Department of AI, Gwangju Institute of Science and Technology (GIST)

Abstract

Recent LLM-based embodied agents have advanced in high-level task planning. Yet, they frequently suffer from hallucinations and brittle reasoning due to passive observation—either omitting fine-grained attributes or overwhelming the planner with irrelevant information. We propose SCENEQUERYPLANNER, a memory–environment dual-grounded agent that integrates active information seeking into the planning loop by decomposing instructions and selectively executing environmental interactions or structured queries over a dynamic 3D scene graph to retrieve task-relevant information on demand. By treating information acquisition as a first-class planning action, our approach enables explicit verification and supports reliable long-horizon reasoning over object attributes. For rigorous evaluation, we introduce ATTPLAN-BENCH, a benchmark of multi-room, long-horizon household tasks requiring attribute-aware reasoning. Experiments demonstrate that our dual-grounding strategy outperforms passive baselines, yielding more stable and scalable reasoning across diverse tasks. Our code is available at <https://github.com/Choi-JaeWoo/SceneQueryPlanner>

1 Introduction

Grounding natural language in physical environments has become one of the core challenges in computational linguistics (Ichter et al., 2023; Driess et al., 2023)—how do we enable systems to bridge the gap between symbolic language expressions and executable actions? Consider the instruction “Place the wooden items from the sink into a box to prevent them from getting wet.” Successfully executing this task requires more than surface-level language understanding—it demands semantic parsing of implicit constraints (material

properties, spatial relations), and structured reasoning over spatial and temporal dependencies.

Despite recent advances, current large language model (LLM)-based agents face a crucial dilemma: they either (1) process exhaustive descriptions of all objects, relations, and hierarchies—leading to context dilution, computational inefficiency, and increased hallucination risk (Liu et al., 2024)—or (2) rely on passive retrieval mechanisms that provide fragmented knowledge (Rana et al., 2023; Honerkamp et al., 2024)—potentially reasoning over incomplete or irrelevant information. Rather than meticulously seeking the specific information to resolve ambiguities or verify constraints, these systems make decisions within constrained contexts. Furthermore, conventional benchmarks provide only flat lists of detected objects, thereby lacking the fine-grained attribute information needed for nuanced semantic decisions.

We formulate active information-seeking, in which agents dynamically populate knowledge through environmental interaction and strategically extract information—treating knowledge acquisition as a first-class action and learning when and what to query. Human intelligence is inherently active and goal-directed (Gibson, 1966): rather than passively absorbing all information, we strategically seek specific details to resolve ambiguities and verify assumptions. Inspired by this cognitive principle, we propose SCENEQUERYPLANNER, a memory–environment dual-grounded agent unifying reasoning, acting, and structured memory querying within a single policy, grounding each step in verified facts (Fig. 1).

Specifically, SCENEQUERYPLANNER combines two complementary grounding mechanisms. Through environment grounding, the planner obtains partial observations for immediate context and local scene graph fragments and dynamically updates a hierarchical 3D scene graph (3DSG) memory. Through memory grounding, it actively

*Equal contribution.

†Corresponding author.

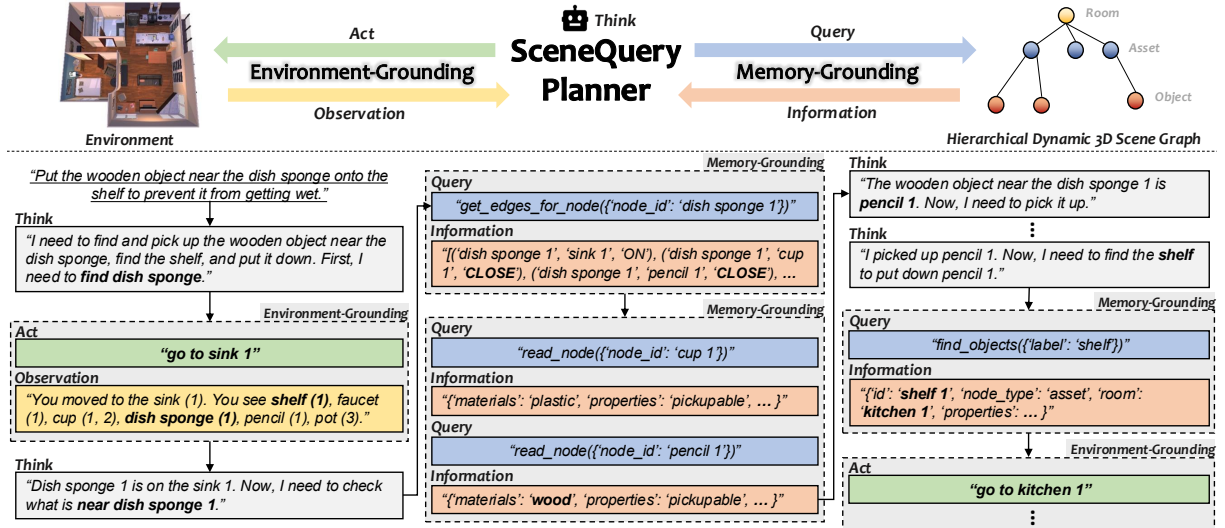


Figure 1: Overview of SCENEQUERYPLANNER. SCENEQUERYPLANNER maintains a dynamic 3D scene graph as a memory and actively issues structured queries for relevant information retrieval.

formulates structured queries to extract targeted information—specific objects, spatial relations, or fine-grained attributes required for the current step—rather than passively consuming serialized memory. This selective retrieval enables attribute-aware reasoning and mitigates hallucinations from unverified assumptions, while each query serves as an interpretable semantic bridge between language and structured knowledge.

Another key contribution of our work is ATTPLAN-BENCH: a comprehensive benchmark designed to evaluate structured language understanding in complex, multi-room environments where success requires (1) parsing instructions with implicit attribute and relational constraints, (2) active knowledge seeking across hierarchical scene representations, and (3) dynamic re-planning in response to environmental changes. Compared to recent benchmarks that scale to multi-room environments but remain largely centered on transporting items with object-centric instructions, ATTPLAN-BENCH extends long-horizon household tasks with richer state transitions (ATTPLAN-BASE) and, critically, introduces attribute-centric tasks where goals are specified through object properties such as material and spatial proximity (ATTPLAN-ATTR).

In summary, our contributions are as follows:

- We propose SCENEQUERYPLANNER, which elevates information acquisition to a first-class planning action, enabling the agent to learn when and what to verify from task experience.
- We introduce ATTPLAN-BENCH: ATTPLAN-BASE for long-horizon household tasks and

ATTPLAN-ATTR for attribute-centric tasks.

- Experiments show state-of-the-art results on both ATTPLAN-BENCH and the external WAH-NL benchmark. We publicly release our code and benchmark.¹

2 Related Work

LLM and VLM-based Embodied Agents. Contemporary embodied agents increasingly leverage LLMs for high-level planning from language instructions, employing strategies such as affordance filtering (Ichter et al., 2023), feedback loops (Huang et al., 2023), self-refinement (Shinn et al., 2023; Sun et al., 2023), tree-based search (Hao et al., 2023; Zhao et al., 2023; Choi et al., 2026), and trajectory learning (Xiong et al., 2024; Qiao et al., 2024). However, these planners predominantly rely on passive observations—typically flat object lists from detectors—that lack the fine-grained attribute information (e.g., material, state) required for complex tasks. While vision-language model (VLM) agents process visual inputs directly, they often exhibit weaker long-horizon reasoning compared to LLM agents (Su et al., 2024; Yang et al., 2025).

LLM Agents with External Memory. To ground LLM agents in household environments, 3DSG-based memory systems are increasingly adopted (Kim et al., 2020; Hughes et al., 2022; Gu et al., 2024; Yan et al., 2025). SAYPLAN (Rana et al., 2023) conducts semantic search over a pre-built

¹<https://github.com/Choi-JaeWoo/SceneQueryPlanner>

static 3DSG to extract task-relevant subgraphs, while MOMA-LLM (Honerkamp et al., 2024) maintains dynamic graphs but inserts rule-based textual descriptions into the prompt. However, these methods passively serialize memory contents into prompts, potentially including task-relevant attributes but burying them among irrelevant details rather than selectively retrieving what the current step requires. Recent approaches further select task-relevant information via embedding-based subgraph retrieval (Booker et al., 2024) or LLM-based filtering of irrelevant objects (Mohammadi et al., 2025). In both cases, however, these operations occur outside the planner’s action space, and the planner passively consumes the result. In NLP, knowledge graph reasoning (Sun et al., 2024; Jiang et al., 2023) has shown that LLMs can actively navigate graphs to extract precise information. SCENEQUERYPLANNER adapts this query-driven paradigm to embodied agents, enabling structured queries over a dynamic 3DSG to actively retrieve task-relevant facts from evolving environments on demand.

Benchmarks for Embodied Agents. Embodied benchmarks have evolved from early instruction-following datasets (Shridhar et al., 2020; Puig et al., 2021) to large-scale multi-room environments, expanding task diversity and realism (Li et al., 2023; Choi et al., 2024; Chang et al., 2025). However, even with large-scale procedural environments such as ProcTHOR (Deitke et al., 2022), existing benchmarks rarely require fine-grained attribute reasoning (e.g., material, state) combined with dynamic information acquisition in long-horizon tasks. Our ATPLAN-BENCH addresses this limitation by introducing multi-room tasks that explicitly demand dynamic, attribute-centric reasoning.

3 SCENEQUERYPLANNER

3.1 Problem Formulation

We consider an embodied agent that follows a natural-language instruction g (e.g., “prepare breakfast using only ripe fruit and clean dishes”) in a partially observable environment $\mathcal{E}$. To enable grounded planning, we define the agent’s action space as $\mathcal{A} = \mathcal{A}_{\text{env}} \cup \mathcal{A}_{\text{rea}} \cup \mathcal{A}_{\text{qry}}$: physical environmental actions ($\mathcal{A}_{\text{env}}$), internal reasoning actions ($\mathcal{A}_{\text{rea}}$), and active memory querying actions ($\mathcal{A}_{\text{qry}}$). At each time step t , the agent selects an action $a_t \in \mathcal{A}$ conditioned on the decision history $h_t = (o_1, a_1, o_2, \dots, a_{t-1}, o_t)$. Each observation

o_t represents the textual information obtained after executing the preceding action, derived from either environmental perception or memory retrieval.

3.2 Dynamic 3D Scene Graph Memory

To support context-aware and attribute-related reasoning, the agent maintains a structured working memory in the form of a dynamic hierarchical 3DSG (Kim et al., 2020; Xu et al., 2025). At time t , memory is a global graph $G_t = (N_t, E_t)$, where nodes N_t and edges E_t encode accumulated environmental knowledge. The graph evolves over time and is maintained as a queryable belief state.

Hierarchy & Attributes. We adopt a three-layer hierarchy following Rana et al. (2023): object nodes $\mathcal{N}_t^1$ (e.g., mug, apple), asset nodes $\mathcal{N}_t^2$ (e.g., countertop, fridge), and room nodes $\mathcal{N}_t^3$ (e.g., kitchen, bedroom). Each node $n \in N_t$ stores fine-grained attributes such as semantic label, type, dynamic state (e.g., CLEAN, OPEN), and 3D position. Edges $e \in E_t$ represent spatial relations from n_i to n_j as ordered triplets (n_i, n_j, r) , where $r \in \{\text{INSIDE, ON, CLOSE}\}$.

Dynamic Update. Unlike static approaches, our memory evolves via incremental fusion—reasoning and querying operate over the most recent representation of the environment. At each time step t , a newly perceived local scene graph o_{t+1}^{graph} is merged into the global belief state G_t through an update operator (Honerkamp et al., 2024): $G_{t+1} = \text{Update}(G_t, o_{t+1}^{\text{graph}})$. This mechanism supports not only adding newly discovered nodes but also updating attributes of existing nodes (e.g., CLEAN $\rightarrow$ DIRTY) to reflect environmental changes (see Appendix G for full details).

3.3 The SCENEQUERYPLANNER Framework

Our approach is a dual-grounded decision-making loop that interleaves reasoning, environment interaction, and memory retrieval (Fig. 1). At each step t , the agent samples an action $a_t \sim \pi(\cdot \mid g, h_t)$ from an expanded action space $\mathcal{A} = \mathcal{A}_{\text{rea}} \cup \mathcal{A}_{\text{env}} \cup \mathcal{A}_{\text{qry}}$, explicitly modeling information acquisition as part of the policy and enabling the agent to decide both *what* to do and *what it needs to know*, in contrast to prior agents that passively consume observations or static memory.

Reasoning ($\mathcal{A}_{\text{rea}}$) generates free-form thoughts to decompose goals or analyze current states (e.g., “I need to check if the apple is sliced”). This step enables the agent to plan logically before committing to physical or querying actions—explicitly formu-

lating hypotheses that can later be verified through grounding actions.

Environment Grounding ($\mathcal{A}_{\text{env}}$) executes physical actions, after which the perception module converts raw sensory inputs into two modalities: a textual observation o_{t+1}^{text} appended to the history h_{t+1} for immediate context, and a local scene graph o_{t+1}^{graph} fused into the global memory to maintain an up-to-date world model. This continual grounding prevents drift inherent in purely prompt-based agents without feedback.

Memory Grounding ($\mathcal{A}_{\text{qry}}$) actively formulates structured queries to retrieve task-relevant information from G_t (Sec. 3.4). This action acquires missing attributes or relations required for reasoning. Retrieved information is returned as a textual answer o_{t+1}^{answer} and appended to the decision history, grounding decisions in verified facts rather than in implicit context. Unlike prior approaches that serialize memory into prompts for passive consumption, our framework enables selective probing of structured memory along hierarchical, relational, and attribute axes—integrating information retrieval directly into planning.

Taken together, the three action modes establish a new planning paradigm in which reasoning, acting, and information seeking are jointly optimized. By explicitly modeling knowledge acquisition, SCENEQUERYPLANNER mitigates the risk of irreversible decisions based on unverified assumptions—a critical limitation of passive LLM agents.

3.4 Active Structured Querying

To support memory grounding and address the attribute blindness of standard LLM agents, we design a lightweight structured query interface that enables the agent to actively retrieve and verify specific facts from the global memory G_t , rather than relying on passive retrieval of broad subgraphs.

Each node $n \in N_t$ is represented as an attribute dictionary and identified by a unique node_id (e.g., fridge 1), while each edge $e \in E_t$ is encoded as a tuple (e.g., (bowl 1, desk 1, ON)). The interface provides the following functions:

- `find_objects(predicate)`: Retrieves a list of node dictionaries given predicate (e.g., `find_objects({'label': 'mug', 'state': 'clean'})` returns all clean mugs)—enabling the agent to identify specific targets or filter candidates using fine-grained attributes beyond flat object lists.
- `get_child_nodes(node_id)`: Returns a list of

node dictionaries corresponding to the children of the specified node in the hierarchy. This allows the agent to inspect the detailed contents of a container or room (e.g., objects inside fridge 1).

- `get_edges_for_node(node_id)`: Returns a list of edges adjacent to the target node, (e.g., (bread 1, countertop 1, ON)), enabling the agent to inspect spatial relations such as containment or proximity for relational reasoning.
- `read_node(node_id)`: Returns the full attribute dictionary of a node. This is used to inspect detailed properties such as checking if a device is toggled on or off.

In contrast to passive memory retrieval, our interface enables information selection and compositional reasoning along orthogonal axes: hierarchical (`get_child_nodes`), relational (`get_edges_for_node`), and attribute-level (`read_node`).

Query Execution & Feedback Loop. By composing these primitives, SCENEQUERYPLANNER issues structured queries to retrieve task-relevant information, where a query action $a_t \in \mathcal{A}_{\text{qry}}$ reads from G_t and returns a textual answer o_{t+1}^{answer} appended to the history h_{t+1} . While inspired by Knowledge-Graph-augmented LLMs (Jiang et al., 2023; Chen et al., 2024), our approach differs fundamentally: queries operate over a *dynamic*, partially observed world model rather than a knowledge graph. This distinction enables a continuous feedback loop where the agent tests hypotheses against the evolving environment state, enforcing a verification-first paradigm that prevents cascading errors from unverified assumptions.

3.5 Learning Active Information Seeking

The expanded action space defines what the agent can do; we now describe how it acquires effective information-seeking strategies from task experience. We first bootstrap a pool of successful planning trajectories from a training set: starting from a small set of human-collected seeds, we run the planner on the training split and retain only trajectories that fully complete the task (Appendix A.2). At each step t , the agent constructs a prompt by concatenating: (1) a system prompt P_{sys} defining the action space $\mathcal{A}$ and query grammar (Appendix A.1); (2) few-shot examples P_{ex} retrieved from the pool via semantic similarity (Reimers and Gurevych, 2019) (Appendix A.3); (3) the instruction g ; and (4) the current history h_t . The next action is sampled from the language

model as: $a_t \sim p_{\text{LLM}}(\cdot \mid P_{\text{sys}}, P_{\text{ex}}, g, h_t)$. By conditioning on trajectories that demonstrate successful query patterns for similar tasks, the agent transfers information-seeking strategies—when to verify and what to query—to novel instructions.

The same trajectory pool also serves as supervised data for LoRA fine-tuning (Hu et al., 2022), which optimizes a next-action prediction objective while masking instruction and observation tokens so that all action types—reasoning, acting, and querying—contribute to the loss (Appendix C), yielding further gains over ICL (Sec. 5.3). ICL is used as the default setting unless otherwise noted.

4 ATTPLAN-BENCH

We introduce ATTPLAN-BENCH, a benchmark for evaluating goal-driven and attribute-aware embodied planning in realistic household environments. Built on ProcTHOR-10K (Deitke et al., 2022) and AI2-THOR (Kolve et al., 2017), it leverages a rich action space and detailed object-attribute annotations to support both standard and attribute-centric household tasks involving spatial relations, material semantics, and affordances. The benchmark comprises two complementary subsets: ATTPLAN-BASE, which extends ALFRED-style tasks (Shridhar et al., 2020) into long-horizon multi-room settings, and ATTPLAN-ATTR, which focuses on instructions requiring attribute-aware reasoning.

4.1 Dataset Generation Pipeline

We construct a large-scale dataset using a semi-automated, LLM-guided generation pipeline (Chang et al., 2025) (Fig. 2) using the LLaMA 3.3 70B instruction-tuned model (Grattafiori et al., 2024). Our pipeline has four stages: (1) task type definition, (2) executable task generation, (3) instruction paraphrasing, and (4) task goal generation.

Task type definition. We define ten task types covering diverse household activities. Using detailed scene specifications (room layouts, object instances, receptacles, and attribute annotations), we prompt an LLM to generate candidate instructions from the 1,000 environments in the ProcTHOR-10K test split. Human annotators then cluster these instructions into ten ALFRED-style task types spanning varying levels of difficulty, from simple object relocation to tasks involving state transitions (e.g., heating) and sequential dependencies. These task types constitute the basis of both ATTPLAN-

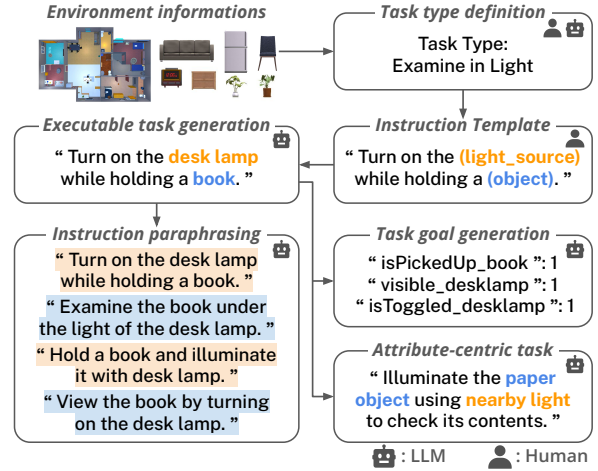


Figure 2: Dataset generation pipeline. We construct the dataset via a semi-automated, LLM-powered pipeline.

BASE and ATTPLAN-ATTR, enabling systematic evaluation across different planning complexities. Additional descriptions, examples, and prompts are provided in Appendix B.

Executable task generation. For each task type, we specify a natural language instruction template and provide the LLM with environment-specific context to generate grounded, executable instructions (e.g., template: “Collect (a/an) (two or three objects) and place them (on/in) the (receptacle)” → instance: “Collect a laptop and a credit card and place them on the armchair.”). The authors curate all generated instructions without external annotators to ensure clarity, feasibility, and attribute-constraint alignment. Executability is guaranteed via an automated oracle check: every task gets simulated in AI2-THOR, and any instance found to be physically inconsistent or infeasible is discarded. We retain 100 valid instructions per task type, yielding 1,000 task instructions.

Instruction paraphrasing. To ensure linguistic diversity, we paraphrase each instruction into three additional variants using the LLM, resulting in four semantically equivalent but lexically diverse versions. During evaluation, one variant is selected at random to prevent overfitting to fixed syntax.

Task goal generation. Finally, we derive a structured goal representation for each instruction to enable automatic evaluation. For example, the instruction “Make a mug of coffee and cook a potato and an egg, then place everything on the dining table,” is converted into a goal state $\{ \text{‘isCooked_potato’}:1, \text{ ‘isCooked_egg’}:1, \dots \}$ that explicitly encodes the desired object states and spatial arrangements, allowing consistent

Dataset	Planner	LLaMA 3.1 8B		LLaMA 3.1 70B		Qwen 2.5 7B		Qwen 2.5 72B	
		TSR (%)	SSR (%)	TSR (%)	SSR (%)	TSR (%)	SSR (%)	TSR (%)	SSR (%)
WAH-NL	ReAct	5.00	<u>21.22</u>	<u>23.00</u>	<u>42.77</u>	7.00	<u>23.63</u>	<u>21.00</u>	<u>39.37</u>
	SayPlan	2.00	11.10	5.00	13.18	4.00	10.12	19.00	30.55
	MoMa-LLM	0.00	9.73	6.00	22.88	2.00	12.02	8.00	31.00
	SceneQueryPlanner	10.00	29.88	26.00	44.60	11.00	26.90	29.00	45.73
AttPlan-Base	ReAct	34.60	47.74	50.80	<u>59.79</u>	<u>35.20</u>	<u>47.39</u>	<u>52.00</u>	<u>62.33</u>
	SayPlan	<u>38.00</u>	<u>49.68</u>	<u>51.20</u>	59.10	26.80	36.82	19.80	26.02
	MoMa-LLM	17.60	32.77	30.40	47.16	13.40	28.79	30.00	46.77
	SceneQueryPlanner	46.60	57.23	62.20	70.14	44.80	57.57	65.40	72.62
AttPlan-Attr	ReAct	<u>38.80</u>	<u>44.51</u>	<u>46.80</u>	<u>52.24</u>	<u>28.80</u>	<u>35.33</u>	<u>44.00</u>	<u>48.99</u>
	SayPlan	16.40	22.75	18.00	25.78	7.20	12.35	5.60	9.00
	MoMa-LLM	20.80	25.97	37.20	46.41	15.60	22.22	32.00	41.29
	SceneQueryPlanner	54.40	61.45	69.60	74.55	49.60	56.00	68.00	73.65

Table 1: Performance on WAH-NL and ATTPLAN-BENCH. TSR denotes Task Success Rate, and SSR denotes Subgoal Success Rate. The best and second-best results are highlighted in bold and underlined, respectively.

evaluation across planners.

Dataset Split. We split the dataset into 500 training and 500 testing examples. The training set is used for bootstrapped in-context learning or fine-tuning, while the testing set is reserved for evaluation.

4.2 Attribute-Centric Task

In the real world, users often specify goals using attributes such as spatial relations, materials, or affordances rather than object categories alone, e.g., “Place the *glass cup* next to the bread” or “Move the *wooden items* near the sink into the box to keep them dry”—requiring attribute-aware reasoning beyond object-centric commands. However, most existing benchmarks (Shridhar et al., 2020; Puig et al., 2021; Choi et al., 2024) primarily focus on object-centric instructions, overlooking such attribute-aware reasoning.

To address this gap, we construct ATTPLAN-ATTR by augmenting instructions from ATTPLAN-BASE. Given an original instruction (e.g., “Put the pen in the box”) and object attributes (e.g., pen: {material: wood, parent: sink}), an LLM rewrites it into an attribute-aware form (e.g., “Put the wooden object on the sink into the box.”). After filtering unnatural or infeasible generations, we curate 500 instructions with a 250/250 train/test split to explicitly evaluate attribute grounding and relational reasoning.

5 Experiments

5.1 Experimental Settings

Environment and datasets. We evaluate SCENEQUERYPLANNER on two complementary benchmarks: WAH-NL (Puig et al., 2021; Choi et al.,

2024) and our ATTPLAN-BENCH. WAH-NL focuses on multi-room, long-horizon tasks dominated by simple *Pick & Place* operations, where success mainly depends on object localization and transport. In contrast, ATTPLAN-BENCH supports a richer action space and explicitly requires attribute- and relation-conditioned reasoning with multi-step state transitions (e.g., heating, cleaning, toggling)—reflecting real-world household tasks.

Metrics. We report Task Success Rate (TSR) and Subgoal Success Rate (SSR); TSR compares the final state with the goal condition, and SSR measures the ratio of successfully completed subgoals. Further, we measure the average number of decision steps per task (Steps), average input/output tokens per step (I/O Tokens), and total execution time (Time) to assess computational cost.

Baselines. We compare SCENEQUERYPLANNER against three representative approaches: (1) REACT (Yao et al., 2023) relies solely on textual observations without external memory; (2) SAYPLAN (Rana et al., 2023) uses a static full 3DSG and performs semantic search to retrieve task-relevant subgraphs; and (3) MOMA-LLM (Honerkamp et al., 2024) maintains a dynamic 3DSG but passively translates graph states into structured textual observations via rule-bases.

LLM Backbones. We evaluate all planners with LLaMA 3.1 (8B, 70B) (Grattafiori et al., 2024) and Qwen 2.5 (7B, 72B) (Yang et al., 2024), covering two model families at both small and large scales.

5.2 Main Results

Active Querying Outperforms Both Pure Observation and Passive Memory: Table 1 shows

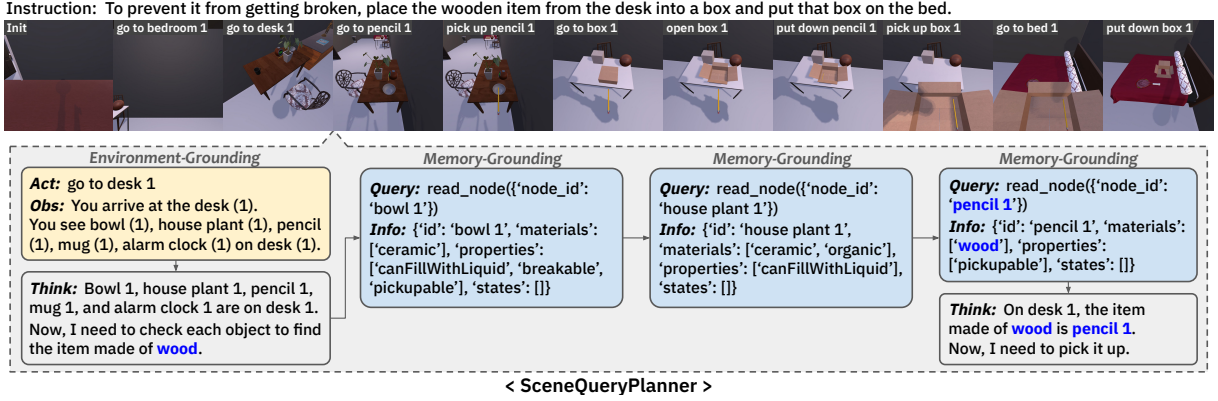


Figure 3: Step-by-step trajectory of SCENEQUERYPLANNER on the ATTPLAN-ATTR task using LLaMA 3.1 70B.

SCENEQUERYPLANNER consistently achieves the highest TSR and SSR across all benchmarks and model scales. SAYPLAN and MOMA-LLM, despite accessing the same 3DSG as SCENEQUERYPLANNER, frequently underperform REACT, which uses no external memory at all—for instance, on ATTPLAN-ATTR with LLaMA 3.1 70B, REACT achieves 46.80% TSR while SAYPLAN and MOMA-LLM reach only 18.00% and 37.20%, respectively. In contrast, SCENEQUERYPLANNER attains 69.60%, substantially outperforming all baselines. This indicates that access to structured memory is insufficient; indiscriminate consumption can even hurt by introducing irrelevant context. What matters is not whether the agent can access information, but whether it retrieves precisely what the current planning step requires.

SCENEQUERYPLANNER Achieves the Strongest Cross-Dataset Stability: SCENEQUERYPLANNER remains robust across datasets, whereas baselines suffer marked performance degradation under distribution shifts. For instance, SAYPLAN attains competitive results on ATTPLAN-BASE yet fails on WAH-NL despite using a large model. In contrast, SCENEQUERYPLANNER maintains stable, top-tier performance on both ATTPLAN-BENCH and external WAH-NL. We attribute this stability to active querying addressing different bottlenecks in each setting: in WAH-NL, it mitigates errors from long interaction histories; in ATTPLAN-BASE, it supports multi-step state tracking; and in ATTPLAN-ATTR, it enables fine-grained attribute verification.

Qualitative Evidence Shows Mechanistic Differences in Reasoning: Fig. 3 shows a representative trajectory from ATTPLAN-ATTR. The task requires identifying a “wooden item.” SCENEQUERYPLANNER resolves ambiguity by iteratively querying

Dataset	Learning Strategy	LLaMA 3.1 8B		Qwen 2.5 7B	
		TSR	SSR	TSR	SSR
AttPlan-Base	ICL	46.60	57.23	44.80	57.57
	LoRA	59.40	69.03	61.40	70.76
AttPlan-Attr	ICL	54.40	61.45	49.60	56.00
	LoRA	60.40	68.17	60.80	68.97

Table 2: Comparison between ICL and LoRA fine-tuning on ATTPLAN-BENCH.

the material attributes of candidate objects, correctly distinguishing the wooden pencil from other items. In contrast, an observation-only agent REACT relies on implicit assumptions and incorrectly identifies a ceramic bowl as wooden, leading to cascading errors and task failure (see Fig. 7 and Lists 7–8 in the Appendix F). Beyond error prevention, we also observe emergent recovery: when the agent misses a target during exploration, it issues find_objects to re-locate the item and corrects its plan without an explicit recovery module (see Fig. 8 in Appendix F).

Gains Persist with GPT-4o: SCENEQUERYPLANNER also outperforms REACT with GPT-4o on all three benchmarks, with the largest gain on ATTPLAN-ATTR (+17.2 TSR). As GPT-4o does not expose token log-probabilities, we use free-form generation followed by executable-action mapping; full results and details are provided in Appendix I.

5.3 Learning Strategy Analysis

We compare ICL with parameter-efficient LoRA fine-tuning (Hu et al., 2022) to assess the effectiveness of SCENEQUERYPLANNER across learning paradigms (Table 2). Both methods use the same bootstrapped pool of successful trajectories, isolating the effect of learning strategy. Training configurations are provided in Appendix C.

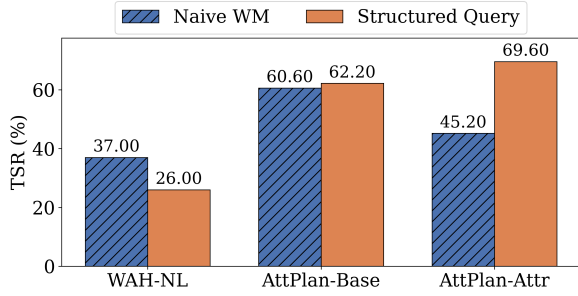


Figure 4: Comparison between SCENEQUERYPLANNER with Naive WM and a structured query interface using LLaMA 3.1 70B.

LoRA fine-tuning consistently outperforms ICL across both subsets and model backbones, demonstrating that active information-seeking strategies can be reliably internalized through parameter-efficient adaptation. The gains are observed not only on ATTPLAN-BASE, where the larger training set (500 trajectories) and unambiguous instructions facilitate procedural learning, but also on ATTPLAN-ATTR (250 trajectories), confirming that the dual-grounding framework benefits from parametric learning even for attribute-centric tasks.

5.4 Interface Analysis

To isolate the impact of structured querying, we compare SCENEQUERYPLANNER with a naive working memory (Naive WM) baseline (Choi et al., 2026) that supports only a single query primitive (recall_location_of). Fig. 4 shows Naive WM outperforms the structured query interface on WAH-NL, where object localization alone suffices, and both variants perform comparably on ATTPLAN-BASE. However, the gap widens substantially on ATTPLAN-ATTR, where the structured query interface achieves a clear advantage through fine-grained attribute and relation verification. These results indicate that embodied planning is constrained not by memory capacity but by the expressiveness of memory access. Full numerical results are provided in Appendix D.

We further analyze query primitive usage across successful trajectories on ATTPLAN-ATTR with Qwen 2.5 72B. Among 1,090 queries, read_node is used most frequently (50.8%), followed by find_objects (33.9%), get_edges_for_node (9.7%), and get_child_nodes (5.6%). All four primitives are used non-trivially, with the dominance of read_node consistent with the attribute-centric nature of ATTPLAN-ATTR. Together with

Planner	Steps	I/O Tokens	Time (ms)
ReAct	28.1	5065.2/11.6	1644.1
SayPlan	-	6258.1/34.5	2812.6
MoMa-LLM	100.6	5748.0/12.7	5901.1
Naive WM	29.4	5057.7/10.6	<u>1662.1</u>
SceneQueryPlanner	36.6	5531.7/ <u>10.7</u>	1706.8

Table 3: Computational complexity comparison on ATTPLAN-BASE, measured using LLaMA 3.1 70B with 4×NVIDIA A100 GPUs.

the Naive WM ablation, this usage pattern supports the practical relevance of the richer query interface rather than suggesting that its additional primitives are largely redundant.

5.5 Computational Efficiency

Table 3 compares the computational costs on ATTPLAN-BASE (Appendix E for other models). We exclude SAYPLAN from the decision-step comparison, as it produces a full plan in a single step.

REACT is the most lightweight baseline, requiring the fewest steps and shortest execution time due to its memory-free design, but this efficiency comes at the cost of weaker reasoning and lower success rates. SCENEQUERYPLANNER incurs modest overhead from structured querying yet remains far more efficient than MOMA-LLM, whose repeated prompt re-encoding leads to hundreds of steps and the highest runtime. Although SAYPLAN plans in one pass, it generates excessively long outputs, inflating token cost. SCENEQUERYPLANNER achieves execution times close to REACT while substantially improving task success—targeted querying offers a balanced efficiency-performance trade-off without significant computational burden.

5.6 Robustness to Perception Noise

To assess robustness under imperfect perception, we conduct noise robustness experiments on ATTPLAN-BENCH by injecting noise into the partial scene graph o_{t+1}^{graph} after each action. Five noise types are applied independently with probability $p \in \{0.1, 0.2, 0.3\}$: (1) node omission, (2) category confusion, (3) attribute flip, (4) edge omission, and (5) edge relation swap. The same protocol is applied to both ATTPLAN-BASE and ATTPLAN-ATTR. The corrupted graph affects both textual observations and, for SCENEQUERYPLANNER, the 3DSG memory—while REACT is exposed only to (1)–(2). Because noise is applied to every eligible element at every step, corruption compounds

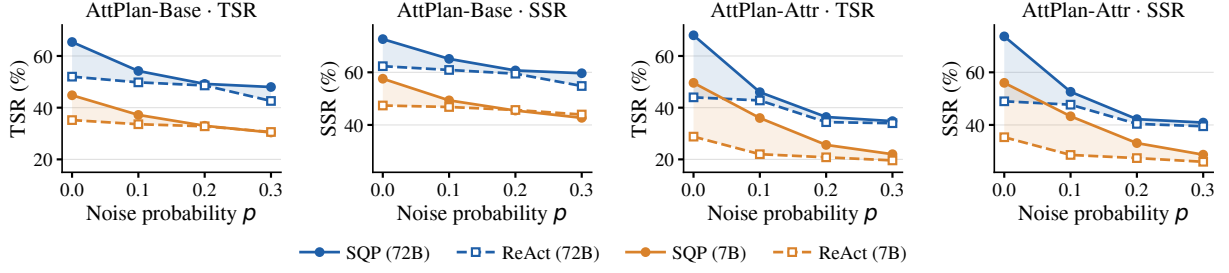


Figure 5: Robustness to perception noise on ATTPLAN-BASE and ATTPLAN-ATTR with Qwen 2.5. Shaded regions indicate the performance gap between SCENEQUERYPLANNER and REACT.

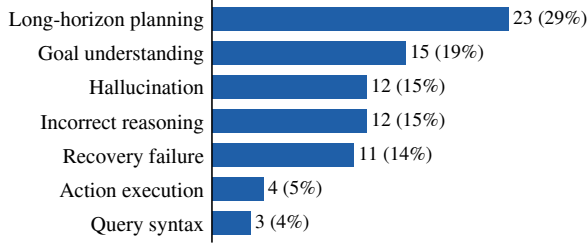


Figure 6: Distribution of primary failure categories across 80 failed trajectories of SCENEQUERYPLANNER on ATTPLAN-ATTR with Qwen 2.5 72B.

throughout the episode. We omit SAYPLAN and MOMA-LLM due to their lower clean performance (Table 1). Details are in Appendix H.

Fig. 5 shows that, on ATTPLAN-BASE, SCENEQUERYPLANNER with the 72B backbone outperforms REACT across all noise levels, including the most severe ($p = 0.3$). With the 7B backbone, however, its advantage largely disappears under stronger corruption. On ATTPLAN-ATTR, in contrast, SCENEQUERYPLANNER consistently outperforms REACT across all noise levels and both backbones. However, the advantage narrows as p increases, since ATTPLAN-ATTR relies on fine-grained attribute information whose corruption directly undermines reliable verification.

5.7 Failure Analysis

To understand the remaining limitations of SCENEQUERYPLANNER, one of the authors manually inspected all 80 failed trajectories on ATTPLAN-ATTR with Qwen 2.5 72B and categorized each into one of seven categories based on its primary cause. Fig. 6 summarizes the resulting distribution.

Long-horizon planning is the largest category (29%), involving incorrect action ordering, incomplete exploration, and subgoal sequencing errors (e.g., attempting to slice without first picking up a knife or revisiting explored rooms while leav-

ing others unexplored). Goal understanding errors (19%) arise from misinterpreting the instruction or declaring completion before all subgoals are satisfied. Hallucination (15%) refers to asserting facts that contradict the agent’s own observations, such as claiming to have found an object that is absent or denying the presence of an observed object. Incorrect reasoning (15%) denotes misinterpreting query results or drawing invalid inferences from correct information. Recovery failures (14%) occur when the agent repeats failed actions instead of adapting its strategy. The remaining failures involve incorrect action execution (5%), where the action does not match the intended reasoning, and incorrect query syntax (4%), such as malformed queries or references to non-existent labels.

Overall, most failures stem from higher-level planning, goal understanding, reasoning, and recovery, while misuse of the query interface is relatively rare. These results suggest that stronger long-horizon planning and failure-aware replanning could further improve the framework.

6 Conclusion

We presented SCENEQUERYPLANNER, a dual-grounded embodied planning framework that treats information acquisition as a first-class action via structured querying over a dynamic 3DSG. By explicitly retrieving task-relevant objects, relations, and attributes, it reduces hallucination and enables reliable long-horizon planning under partial observability. We also introduced ATTPLAN-BENCH for evaluating attribute-aware and relational reasoning in multi-room tasks. Experiments show that active, verification-driven grounding outperforms observation-only and passive-memory baselines while maintaining competitive efficiency. Overall, our results highlight structured and selective information seeking as a key mechanism for robust and generalizable embodied reasoning.

Limitations

First, the effectiveness of SCENEQUERYPLANNER partly depends on upstream perception for dynamic 3DSG. Although the planner actively verifies information through structured queries, inaccuracies in object detection or attribute estimation may still lead to decisions grounded in incorrect premises.

Second, the active information seeking is currently conducted by a pre-defined set of structured query primitives. Future work could explore learning a dynamic query interface or leveraging natural language-to-graph translation to support broader and more flexible investigative capabilities.

Lastly, our evaluation with physics-based simulators enables scalable and reproducible experimentation but does not fully capture the variability of real-world environments—particularly low-level execution and sensor noise. Bridging the gap is an important direction for future work.

Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2024-00336738, Development of Complex Task Planning Technologies for Autonomous Agents, 20%, RS-2022-II220951, Development of Uncertainty-Aware Agents Learning by Asking Questions, 20%, and IITP-2026-RS-2026-25614465, AI Star Fellowship Support Program, 20%), the National Research Council of Science & Technology (NST) grant by the Korea government (MSIT) (No. GTL25042-000, 20%), and the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT)(RS-2026-25510381, 20%)

References

- Meghan Booker, Grayson Byrd, Bethany Kemp, Aurora Schmidt, and Corban Rivera. 2024. [Embodiedrag: Dynamic 3d scene graph retrieval for efficient and scalable robot task planning](#). *Preprint*, arXiv:2410.23968.
- Matthew Chang, Gunjan Chhablani, Alexander Clegg, Mikael Dallaire Cote, Ruta Desai, Michal Hlavac, Vladimir Karashchuk, Jacob Krantz, Roozbeh Motlaghi, Priyam Parashar, Siddharth Patki, Ishita Prasad, Xavier Puig, Akshara Rai, Ram Ramrakhya, Daniel Tran, Joanne Truong, John Turner, Eric Undersander, and Tsung-Yen Yang. 2025. [Partnr: A benchmark for planning and reasoning in embodied multi-agent tasks](#). In *International Conference on Learning Representations*, pages 65205–65268.
- Liyi Chen, Panrong Tong, Zhongming Jin, Ying Sun, Jieping Ye, and Hui Xiong. 2024. [Plan-on-graph: Self-correcting adaptive planning of large language model on knowledge graphs](#). In *Advances in Neural Information Processing Systems*, volume 37, pages 37665–37691. Curran Associates, Inc.
- Jae-Woo Choi, Hyungmin Kim, Hyobin Ong, Youngwoo Yoon, Minsu Jang, Dohyung Kim, and Jaehong Kim. 2026. [Reactree: Hierarchical llm agent trees with control flow for long-horizon task planning](#). In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems*, pages 319–328. International Foundation for Autonomous Agents and Multiagent Systems.
- Jae-Woo Choi, Youngwoo Yoon, Hyobin Ong, Jaehong Kim, and Minsu Jang. 2024. [Lota-bench: Benchmarking language-oriented task planners for embodied agents](#). In *International Conference on Learning Representations*, pages 52718–52744.
- Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Motlaghi. 2022. [Proctor: Large-scale embodied ai using procedural generation](#). In *Advances in Neural Information Processing Systems*, volume 35, pages 5982–5994. Curran Associates, Inc.
- Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, and 3 others. 2023. [PaLM-e: An embodied multimodal language model](#). In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 8469–8488. PMLR.
- James Jerome Gibson. 1966. *The senses considered as perceptual systems*. Houghton Mifflin.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. [The llama 3 herd of models](#). *Preprint*, arXiv:2407.21783.
- Qiao Gu, Ali 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. 2024. [Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning](#). In *2024 IEEE International Con-*

- ference on Robotics and Automation (ICRA)*, pages 5021–5028.
- Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173, Singapore. Association for Computational Linguistics.
- Daniel Honerkamp, Martin Büchner, Fabien Despinoy, Tim Welschhold, and Abhinav Valada. 2024. Language-grounded dynamic scene graphs for interactive object search with mobile manipulation. *IEEE Robotics and Automation Letters*, 9(10):8298–8305.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.
- Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2023. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 1769–1782. PMLR.
- Nathan Hughes, Yun Chang, and Luca Carlone. 2022. Hydra: A Real-time Spatial Perception System for 3D Scene Graph Construction and Optimization. In *Proceedings of Robotics: Science and Systems*, New York City, NY, USA.
- Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, and 399 others. 2024. *Gpt-4o system card*. Preprint, arXiv:2410.21276.
- Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, and 26 others. 2023. Do as i can, not as i say: Grounding language in robotic affordances. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 287–318. PMLR.
- Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023. StructGPT: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9237–9251, Singapore. Association for Computational Linguistics.
- Ue-Hwan Kim, Jin-Man Park, Taek-jin Song, and Jong-Hwan Kim. 2020. 3-d scene graph: A sparse and semantic representation of physical environments for intelligent agents. *IEEE Transactions on Cybernetics*, 50(12):4921–4933.
- Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Gupta, and Ali Farhadi. 2017. *Ai2-thor: An interactive 3d environment for visual ai*. Preprint, arXiv:1712.05474.
- Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, Mona Anvari, Minjune Hwang, Manasi Sharma, Arman Aydin, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R Matthews, and 9 others. 2023. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 80–93. PMLR.
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.
- Microsoft. 2023. Guidance. <https://github.com/guidance-ai/guidance>.
- Mohammad Mohammadi, Daniel Honerkamp, Martin Büchner, Matteo Cassinelli, Tim Welschhold, Fabien Despinoy, Igor Gilitschenski, and Abhinav Valada. 2025. More: Mobile manipulation rearrangement through grounded language reasoning. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1805–1811.
- Xavier Puig, Tianmin Shu, Shuang Li, Zilin Wang, Yuan-Hong Liao, Joshua B. Tenenbaum, Sanja Fidler, and Antonio Torralba. 2021. Watch-and-help: A challenge for social perception and human-ai collaboration. In *International Conference on Learning Representations*.
- Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Jiang, Chengfei Lv, and Huajun Chen. 2024. AutoAct: Automatic agent learning from scratch for QA via self-planning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3003–3021, Bangkok, Thailand. Association for Computational Linguistics.
- Krishan Rana, Jesse Haviland, Sourav Garg, Jad AbouChakra, Ian Reid, and Niko Suenderhauf. 2023. Say-plan: Grounding large language models using 3d scene graphs for scalable robot task planning. In

- Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 23–72. PMLR.
- Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652. Curran Associates, Inc.
- Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10737–10746.
- Ying Su, Zhan Ling, Haochen Shi, Jiayang Cheng, Yauwai Yim, and Yangqiu Song. 2024. ActPlan-1K: Benchmarking the procedural planning ability of visual language models in household activities. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14953–14965, Miami, Florida, USA. Association for Computational Linguistics.
- Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. Adaplaner: Adaptive planning from feedback with language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 58202–58245. Curran Associates, Inc.
- Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *International Conference on Learning Representations*, pages 3868–3898.
- Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024. Watch every step! LLM agent learning via iterative step-level process refinement. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1556–1572, Miami, Florida, USA. Association for Computational Linguistics.
- Wenting Xu, Viorela Ila, Luping Zhou, and Craig T Jin. 2025. Tb-hsu: Hierarchical 3d scene understanding with contextual affordances. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 8960–8968.
- Zhijie Yan, Shufei Li, Zuoxu Wang, Lixiu Wu, Han Wang, Jun Zhu, Lijiang Chen, and Jihong Liu. 2025. Dynamic open-vocabulary 3d scene graphs for long-term language-guided mobile manipulation. *IEEE Robotics and Automation Letters*, 10(5):4252–4259.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 23 others. 2024. [Qwen2.5 technical report](#). *Preprint*, arXiv:2412.15115.
- Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, Heng Ji, Huan Zhang, and Tong Zhang. 2025. EmbodiedBench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 70576–70631. PMLR.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.
- Zirui Zhao, Wee Sun Lee, and David Hsu. 2023. Large language models as commonsense knowledge for large-scale task planning. In *Advances in Neural Information Processing Systems*, volume 36, pages 31967–31987. Curran Associates, Inc.

A Implementation Details

We provide the detailed implementation settings for our proposed SCENEQUERYPLANNER and three baselines: REACT (Yao et al., 2023), SAYPLAN (Rana et al., 2023), and MoMA-LLM (Honerkamp et al., 2024). All planners are implemented in an in-context learning setting without additional fine-tuning. Further implementation details are described below.

A.1 System Prompt

We use system prompts for each planner as shown in Lists 1, 2, 3, and 4. For baselines, we adopt prompts that closely follow those used in their original papers.

Listing 1: System Prompt for SceneQueryPlanner

```
You are an advanced robot with the ability
to think and act. You can perform one
of the following tasks:
1. Think: Use reasoning to satisfy the
current goal condition.
2. Act: Execute a specific action to
accomplish the current goal condition.
You should use one of the actions of
this list: [go to, pick up, put down,
open, close, turn on, turn off, slice,
drop, done, failure]
3. Query: Interact with the 3D scene graph.
You should use one of these queries:
- find_objects({'label': str, 'attrs':
dict})
- get_child_node_names({'parent_node_id':
str, 'child_node_attrs': dict})
- get_edges_for_node({'node_id': str,
'relation': str})
- read_node({'node_id': str, 'attrs': str
or list})

Available attrs: 'toggleable',
'canFillWithLiquid', 'cookable',
'breakable', 'isHeatSource',
'isColdSource', 'sliceable',
'openable', 'pickupable', 'isToggled',
'isFilledWithLiquid', 'isCooked',
'isSliced', 'isOpen', 'isPickedUp',
'isBroken'
Available relations: 'CLOSE', 'ON', 'INSIDE'
```

Listing 2: System Prompt for ReAct

```
You are an advanced robot with the ability
to think and act. You can perform one
of the following tasks:
1. Think: Use reasoning to satisfy the
current goal condition.
2. Act: Execute a specific action to
accomplish the current goal condition.
You should use one of the actions of
this list: [go to, pick up, put down,
open, close, turn on, turn off, slice,
drop, done, failure]
```

Listing 3: System Prompt for SayPlan

```
Agent Role: You are an excellent graph
planning agent. Given a graph
representation of an environment, you
can explore the graph by expanding
nodes to find the items of interest.
You can then use this graph to generate
a step-by-step task plan that the agent
can follow to solve a given instruction.
Environment Functions:
go to: Navigate to the object or room.
pick up: Pick up the object. You need to
use the go to (object_name) command
first to move close to the object
before picking it up. You can only hold
one object at a time.
put down: Place the object currently held
in your hand onto the nearest available
receptacle.
open: Open the object that is currently
closed.
close: Close the object that is currently
opened.
turn on: Turn on the object that is
currently turned off.
turn off: Turn off the object that is
currently turned on.
slice: Slice a cuttable object. You must be
holding a knife or butter knife to
perform this action. When you perform
this action, a new object named sliced
(object_name) will appear in the
environment.
drop: Drop the object currently held in
your hand.
done: Call this when the task is completed.
Environment State:
OPEN: Asset can be closed or kept open
CLOSED: Asset can be open
ON: Asset is currently on
OFF: Asset is currently off
Environment API:
expand_node(<node>): Reveal assets/objects
connected to a room node.
contract_node(<node>): Hide assets/objects,
reducing graph size for memory
constraints.
Output Response Format:
{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
mode: "exploring" OR "planning"
command: {"command_name": Environment API
call
"node_name": node to perform an operation on
"plan": task plan if in planning mode}}
```

Listing 4: System Prompt for MoMa-LLM

```
You are an advanced robot with the ability
to reason and act in a household
environment.

You have the following actions available
that you can use to achieve this task:
```

1. go to: Navigate to the object or room.
 2. pick up: Pick up the object. You need to use the go to (object_name) command first to move close to the object before picking it up. You can only hold one object at a time. When you go to an object, you will be able to observe nearby objects (on or around it).
 3. put down: Place the object currently held in your hand onto the nearest available receptacle.
 4. open: Open the object that is currently closed.
 5. close: Close the object that is currently opened.
 6. turn on: Turn on the object that is currently turned off.
 7. turn off: Turn off the object that is currently turned on.
 8. slice: Slice a cuttable object. You must be holding a knife or butter knife to perform this action. When you perform this action, a new object named sliced (object_name) will appear in the environment.
 9. drop: Drop the object currently held in your hand.
 10. done: Call this when the task is completed.
 11. failure: Call this if you are unable to take any further actions.
- * To find the location of a target object (e.g., pencil), follow this exploration strategy:
- Visit unexplored rooms or unexplored objects in the environment.
 - For each room, check the objects within it by using go to (object) actions.
 - Each time you go to an object (e.g., countertop, cabinet, table), you will observe its nearby items.
 - Use these observations to determine whether the target object is present.
 - Avoid revisiting the same rooms/objects unless necessary.
- * Example - Find a pencil and place it on the countertop:
1. Explore unexplored rooms: Use go to bedroom 1, go to kitchen 2, etc., to enter unexplored rooms.
 2. Explore objects in the room: In each room, use actions like go to cabinet 1, go to desk 1, or go to countertop 1 to examine nearby items. You will observe surrounding objects (e.g., pencil) when you move to one.
 3. If you find the pencil (e.g., pencil 1): First, go near it: go to pencil 1
 4. Then, pick it up: pick up pencil 1
 5. To place the pencil on a countertop (e.g., countertop 1): Make sure you are in the same room as countertop 1. If not, you must first navigate to the room containing it.
 6. Then, move to the countertop: go to countertop 1
 7. Finally, put the pencil down: put down pencil 1

Output Response Format:
 Think: Use reasoning to satisfy the current goal condition.
 Act: Execute a specific action to accomplish the current goal condition.

A.2 In-Context Example Pool Curation

To construct a robust pool of in-context examples, we adopt a bootstrapping strategy. Human experts initially collect a small number of successful trajectories for each task type—specifically, one example per task type for WAH-NL, and two examples each for ATTPLAN-BASE and ATTPLAN-ATTR. These seed trajectories are then used to run all planning methods on the training split of each benchmark using LLaMA 3.1 70B (Grattafiori et al., 2024). Only trajectories that successfully complete the task are retained to form the final pool of in-context examples.

A.3 In-Context Example Retrieval

During inference, relevant in-context examples are retrieved based on sentence embedding similarity between the target task instruction and candidate instructions in the pool, computed using Sentence-BERT (Reimers and Gurevych, 2019). To ensure a fair comparison, we limit the total number of tokens for in-context examples to 5,000 across all methods. An exception is made for SAYPLAN, which employs a semantic search module that extracts task-relevant subgraphs from the full scene graph using expand and collapse operations, resulting in a textual representation that often exceeds the 5K token limit. To accommodate this, we allow SAYPLAN to use only the single most relevant in-context example.

A.4 Experimental Settings

Maximum Decision Steps. All planners, except SAYPLAN, are allowed a maximum of 200 decision steps per task across all benchmarks: WAH-NL, ATTPLAN-BASE, and ATTPLAN-ATTR. SAYPLAN consists of two separate processes—semantic search and iterative replanning—each involving structured decision-making in the form of chain-of-thought, reasoning, and command (for semantic search) or plan (for iterative replanning), as described in the original paper. Due to the high token consumption of semantic search, we allocate a maximum of 30 decision steps for this phase. For iterative replanning, which generates a complete

plan in a single decision rather than step-by-step actions, we set the maximum decision steps to 60, which is sufficient despite being lower than the 200-step budget used for other methods.

LLM Generation Setup. To reduce latency, minimize tokenization artifacts, and enable caching, we employ Microsoft’s Guidance framework (Microsoft, 2023). For environment action generation, we implement the LLM agent such that it selects the skill with the highest likelihood from the currently executable skill set. For reasoning and structured query generation, we use free-form text generation with a temperature of 0.0.

B More Details on ATTPLAN-BENCH

We introduce ATTPLAN-BENCH, a benchmark for evaluating goal-driven and attribute-aware embodied planning in realistic household environments. Unlike prior benchmarks that predominantly emphasize object-centric instructions or short-horizon execution, ATTPLAN-BENCH is designed to stress-test long-horizon planning under partial observability, where success often requires resolving implicit constraints over object attributes (e.g., material, state) and relations (e.g., proximity, containment).

ATTPLAN-BENCH comprises two complementary subsets. ATTPLAN-BASE extends ALFRED-style tasks into long-horizon, multi-room settings that require compositional navigation and manipulation over extended trajectories. ATTPLAN-ATTR targets instructions that explicitly demand attribute-centric grounding, where goals are specified through object properties (e.g., “wooden item,” “toggleable object,” “clean dish”) rather than instance names. The dataset is constructed via a semi-automated generation pipeline, which enables scalable task creation while maintaining physical feasibility through validation and filtering.

Table 4 presents the instruction templates for the ten task types used to instantiate ATTPLAN-BASE. Table 5 summarizes representative example instructions and their structured goal conditions across both ATTPLAN-BASE and ATTPLAN-ATTR, illustrating how tasks require reasoning over object states (e.g., isCooked, isClean, isToggled) and spatial outcomes (e.g., on, in, visible).

Prompts for data generation. To clarify how task instances were generated using LLMs and to support reproducibility, we provide the exact prompts employed in our data generation pipeline. The prompt for *task type definition* is

provided in List 5, while the prompt for generating *environment-grounded executable task instances* is presented in List 6. These prompts show how structured environment descriptions (rooms, objects, receptacles, and attribute annotations) are transformed into natural language instructions that are grounded in the current scene and executable by the embodied agent.

Listing 5: Prompt for Task Type Definition

```
# System Prompt
You are a helpful assistant embedded in a household robot. You receive a structured description of the environment, including:
- The names of the rooms in the current environment,
- A list of objects, each with its type, attributes (e.g., 'toggleable', 'canFillWithLiquid', 'cookable', 'breakable', 'isHeatSource', 'isColdSource', 'sliceable', 'openable', 'pickupable', 'salientMaterials'), and its location (parentReceptacles),
- A list of receptacles, each with its type, attributes, and what objects they contain (receptacleObjectIds).

Your task is to generate natural, realistic, and physically possible human instructions that could be given to a household robot in this specific environment.

Instructions should be:
- Grounded in the current environment (only use objects and receptacles listed),
- Feasible based on the physical constraints (e.g., pickupable == true),
- Contextually appropriate (e.g., putting food in a fridge, throwing trash in a bin),
- Expressed as a single clear sentence in natural spoken English, like something a person would say to a robot assistant,
- Only refer to object types, not instance IDs.

* Only return one instruction per environment. Do not explain. Do not list multiple instructions.

### Input:
Room information:
{rooms}

Object information:
{objects}

Receptacle information:
{receptacles}

### Output format:
Generate one valid instruction that can be executed in the current environment.
```

Task Type	Instruction Template
Pick & Place Single	Pick up (a/an) (object) and put it (on/in) the (receptacle).
Pick & Place Multiple	Collect (a/an) (two or three objects) and place them (on/in) the (receptacle).
Pick & Place Implicit	Move all the objects from the (receptacle 1) to the (receptacle 2).
Examine in Light	Turn on the (light_source) while holding (a/an) (object).
Heat & Place	Heat (a/an) (object) and place it (on/in) the (receptacle).
Cool & Place	Place a cold (object) (on/in) the (receptacle).
Clean & Place	Clean the dirty (object) and place it (on/in) the (receptacle).
Stack & Place	Place (a/an) (container) containing (a/an) (object) (on/in) the (receptacle).
Turn Off	Toggle off all (objects) that are currently on.
Cook & Place	Use the coffee machine to brew coffee into (a/an) (container), then place the coffee (on/in) the (receptacle). Cook (a/an) (object) using a microwave and then place it (on/in) the (receptacle). Toast a slice of bread using a toaster and then place it (on/in) the (receptacle).

Table 4: Instruction templates for each task type.

```
# User Prompt
Now, generate an instruction that is
    executable in the current environment,
    using the provided information and
    following the specified output format.
```

Listing 6: Prompt for Executable Task Generation (Cool & Place task)

```
# System Prompt
You are a system that adapts user
    instructions to be executable by a
    household robot, specifically for
    cool-then-place tasks, based on the
    specific objects and receptacles
    available in the environment.

Do not explain or elaborate. Your output
    must be a valid, well-formed
    instruction string only.

Given the list of object and receptacle
    information from a simulated
    environment, your task is to generate a
    cool-then-place instruction that:
- Uses only objects and receptacles that
    are actually present in the provided
    object or receptacle information.
- Puts the object into or onto a target
    that has "receptacle": true.
- The instruction must be semantically
    appropriate based on real-world
    affordances between the object and the
    cool source.
- The generated instruction must start with
    a capital letter, and the rest of the
    sentence must be in lowercase.
- Use only the objectType field, not the
    object name.

### Input:
Cool source information:
{is_CoolSource}
```

```
Object information:
{objects}

Receptacle information:
{receptacles}

### Output format:
Generate one revised instruction based on
    the following template:
Place a cold (object) (on/in) the
    (receptacle).

# User Prompt
Now, generate a revised instruction that is
    executable in the current environment,
    using the provided information and
    following the specified output format.
```

C LoRA Fine-Tuning Details

For fine-tuning, we adopt a standard LoRA configuration (Hu et al., 2022) that inserts low-rank adapters into attention (q/k/v/o) and MLP layers, with rank 16, scaling factor 32, and dropout 0.1. Each training example is constructed by concatenating instructions and target outputs while masking instruction and observation tokens so that only action predictions—encompassing reasoning, environment actions, and structured queries—contribute to the loss. The model is trained with a sequence length of 4096, batch size 1 with gradient accumulation, and a learning rate of 5×10^{-5} ; during inference, only the adapter weights are retained, enabling lightweight deployment.

Task Type	Instructions	Task Goal
AttPlan-Base (Pick & Place Single)	Slice an apple and put it in the bowl. Put a sliced apple in the bowl. Take an apple, slice it, and put it in the bowl.	"in_apple_bowl": 1, "isSliced_apple": 1
AttPlan-Base (Pick & Place Multiple)	Collect a pencil, a pen, and a book and place them on the countertop. Gather a pencil, a pen, and a book, and set them on the countertop. Place a pencil, a pen, and a book side by side on the countertop.	"on_pencil_countertop": 1, "on_pen_countertop": 1, "on_book_countertop": 1
AttPlan-Base (Pick & Place Implicit)	Move all the objects from the side table to the dining table. Remove everything from the side table and put it on the dining table. Clear off the side table by moving all objects to the dining table.	"empty_sidetable": 1, "moveTo_diningtable": 1
AttPlan-Base (Examine in Light)	Examine the book under the floor lamp's light. Hold a book and turn on the floor lamp to see it clearly. View the book by the illumination of the floor lamp.	"isToggled_floorlamp": 1, "isPickedUp_book": 1, "visible_floorlamp": 1
AttPlan-Base (Heat & Place)	Heat a tomato and place it on the dining table. Place a heated tomato onto the dining table. Put the tomato on the dining table after it has been heated.	"temperatureHot_tomato": 1, "on_tomato_diningtable": 1, "on_temperatureHot_tomato_diningtable": 1
AttPlan-Base (Cool & Place)	Place a cold bottle on the countertop. Put the bottle in the fridge first, then set it on the countertop. Chill the bottle in the fridge, then transfer it to the countertop.	"temperatureCold_bottle": 1, "on_bottle_countertop": 1, "on_temperatureCold_bottle_countertop": 1
AttPlan-Base (Clean & Place)	Clean the cloth and place it in the box. Wash the cloth at the sink basin and then put it in the box. Make sure the cloth is clean before you place it in the box.	"isClean_cloth": 1, "in_cloth_box": 1, "in_isClean_cloth_box": 1
AttPlan-Base (Stack & Place)	Place a plate containing an egg on the dining table. Put a plate with an egg on it onto the dining table. Set a plate holding an egg onto the dining table.	"stack_egg_plate_diningtable": 1, "on_egg_plate": 1, "on_plate_diningtable": 1
AttPlan-Base (Turn Off)	Toggle off all televisions and floor lamps that are currently on. Switch off any floor lamps and televisions that are currently in use. Make sure all televisions and floor lamps have been turned off.	"toggleOff_television": 1, "toggleOff_floorlamp": 1
AttPlan-Base (Cook & Place)	Use the coffee machine to brew coffee into a mug and toast a slice of bread using a toaster, then place the coffee and toast on the countertop. Brew coffee into a mug and toast some bread, then put the coffee and toast on the countertop. Make a mug of coffee using the coffee machine and a piece of toast with the toaster, then set them down on the countertop.	"isCooked_breadsliced": 1, "fillLiquid_coffee_mug": 1, "on_breadsliced_countertop": 1, "on_mug_countertop": 1
AttPlan-Attr	Move the wooden items near the sink to the countertop to prevent them from getting wet.	"on_pencil_countertop": 1
AttPlan-Attr	To make a salad, slice all sliceable items in the fridge and place them in the bowl.	"in_apple_bowl": 1, "in_tomato_bowl": 1, "isSliced_apple": 1, "isSliced_tomato": 1
AttPlan-Attr	Use the desk lamp to illuminate the paper on the dresser and read what it says.	"isToggled_desklamp": 1, "isPickedUp_newspaper": 1, "visible_desklamp": 1
AttPlan-Attr	Use a ceramic item from the desk to make coffee with the coffee machine, and serve it on the countertop.	"fillLiquid_coffee_mug": 1, "on_mug_countertop": 1

Table 5: Overview of task types in ATTPLAN-BENCH with sample instructions and goal conditions.

D Interface Ablation

This section provides a detailed ablation analysis to isolate the contribution of the structured query interface in SCENEQUERYPLANNER. While the full model actively issues attribute-, relation-, and hierarchy-aware queries over a dynamic 3D scene graph, we compare against a *Naive Working Memory (Naive WM)* variant that only supports a single query primitive for object localization (i.e., `recall_location_of`), following prior work (Choi et al., 2026).

Experimental setup. Both variants share the same environment grounding, perception module, in-context learning setup, and maximum decision budget. The only difference lies in the expressiveness of memory access: Naive WM can retrieve object locations but cannot inspect attributes (e.g., material, state) or relations (e.g., CLOSE, INSIDE). We evaluate both planners on WAH-NL, ATTPLAN-BASE, and ATTPLAN-ATTR using identical prompts and model backbones.

Results and analysis. Table 6 reports the full numerical results. On WAH-NL, Naive WM performs comparably to the structured query interface, and slightly better. This is expected, as WAH-NL is dominated by long-horizon *Pick & Place* tasks where object localization is the primary bottleneck and fine-grained attribute reasoning is rarely required.

In contrast, the performance gap becomes substantial on ATTPLAN-BENCH. On ATTPLAN-BASE, the structured query interface consistently improves both Task Success Rate (TSR) and Subgoal Success Rate (SSR) across all model sizes. The gap further widens on ATTPLAN-ATTR, where instructions explicitly specify goals through attributes and relational constraints. Without the ability to interrogate attributes or relations, Naive WM frequently commits early misidentifications that cascade into irreversible failures.

These results demonstrate that embodied planning performance is not limited by the amount of stored memory, but by the expressiveness of memory access. By enabling selective, structured queries over attributes and relations, SCENEQUERYPLANNER transforms the scene graph from a passive record into an actionable reasoning substrate, which is essential for reliable planning under realistic task complexity and attribute ambiguity.

E Computational Efficiency

We provide a detailed analysis of the computational cost of SCENEQUERYPLANNER in comparison to prior embodied agents. We report (1) the average number of decision steps per episode, (2) the average number of input/output tokens processed by the language model, and (3) the total execution time.

Measurement protocol. All measurements are conducted on ATTPLAN-BASE. Experiments with LLaMA 3.1 70B and Qwen 2.5 72B are run on 4×NVIDIA A100 GPUs, while smaller models (LLaMA 3.1 8B, Qwen 2.5 7B) are evaluated on 8×NVIDIA A30 GPUs. To ensure fairness, all planners (except SAYPLAN) are constrained to a maximum of 200 decision steps.

Comparison with baselines. Table 7 summarizes the results. REACT is the most lightweight baseline, as it issues a single prompt per action without any explicit memory access. However, this efficiency comes at the cost of significantly lower success rates, particularly on attribute-centric tasks. MOMA-LLM, on the other hand, incurs the highest computational overhead due to repeated prompt re-encoding of serialized scene graph states, often resulting in hundreds of decision steps and the longest execution times. SAYPLAN generates a complete plan in a single reasoning pass, but this design leads to disproportionately large output sequences and high token consumption, making it less efficient in terms of language model usage.

Despite introducing an explicit querying mechanism, SCENEQUERYPLANNER maintains a favorable efficiency-performance trade-off. While it requires slightly more decision steps than REACT, its total execution time remains close, and its token usage is substantially lower than that of MOMA-LLM and SAYPLAN. This efficiency stems from *targeted information acquisition*: rather than repeatedly re-encoding the full memory state, the planner retrieves only the specific attributes or relations required at each step.

Overall, these results show that structured querying does not impose prohibitive computational overhead. Instead, it enables grounded and reliable reasoning with moderate additional cost, achieving a balanced trade-off between efficiency and task success that is well-suited for long-horizon, attribute-aware embodied planning.

Dataset	Planner	LLaMA 3.1 8B		LLaMA 3.1 70B		Qwen 2.5 7B		Qwen 2.5 72B	
		TSR (%)	SSR (%)	TSR (%)	SSR (%)	TSR (%)	SSR (%)	TSR (%)	SSR (%)
WAH-NL	Naive WM	13.00	31.88	37.00	54.32	18.00	32.85	34.00	47.78
	Structured Query	10.00	29.88	26.00	44.60	11.00	26.90	29.00	45.73
AttPlan-Base	Naive WM	39.20	49.56	60.60	68.53	40.00	52.23	62.00	70.56
	Structured Query	46.60	57.23	62.20	70.14	44.80	57.57	65.40	72.62
AttPlan-Attr	Naive WM	33.60	41.42	45.20	52.83	28.40	34.79	44.80	51.60
	Structured Query	54.40	61.45	69.60	74.55	49.60	56.00	68.00	73.65

Table 6: Comparison between SCENEQUERYPLANNER with Naive Working Memory (Naive WM) and Structured Query interface.

Planner	LLaMA 3.1 8B				LLaMA 3.1 70B				Qwen 2.5 7B				Qwen 2.5 72B			
	Steps	I/O	Tokens	Time	Steps	I/O	Tokens	Time	Steps	I/O	Tokens	Time	Steps	I/O	Tokens	Time
ReAct	24.0	4940.6	11.6	1457.8	28.1	5065.2	11.6	1644.1	18.9	4683.1	11.5	1467.8	24.5	4660.4	11.9	1884.2
SayPlan	-	5854.6	33.3	1698.1	-	6258.1	34.5	2812.6	-	5566.4	25.4	1344.2	-	5647.9	27.2	2397.4
MoMa-LLM	110.9	5765.0	11.9	2434.6	100.6	5748.0	12.7	5901.1	128.5	5690.2	13.6	2733.8	99.9	5657.8	13.0	8163.5
Naive WM	36.6	5139.6	10.5	1244.3	29.4	5057.7	10.6	1662.1	27.0	4838.3	10.2	1217.3	25.7	4930.6	10.7	1781.8
Ours	33.5	5448.5	11.0	1337.1	36.6	5531.7	10.7	1706.8	28.2	5319.3	11.7	1679.6	34.9	5435.2	10.5	2765.9

Table 7: Computational complexity comparison. Measurements for LLaMA 3.1 70B and Qwen 2.5 72B were conducted using 4xNVIDIA A100 GPUs, while LLaMA 3.1 8B and Qwen 2.5 7B were measured using 8xNVIDIA A30 GPUs.

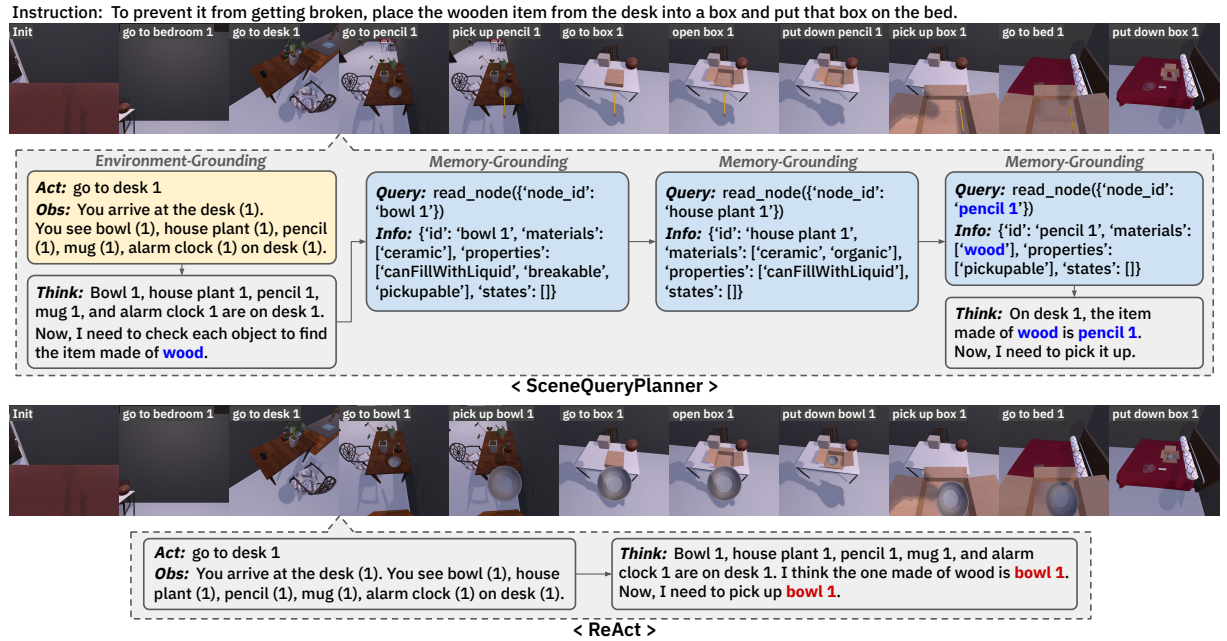


Figure 7: Comparison of SCENEQUERYPLANNER and REACT on the ATTPLAN-ATTR task using LLaMA 3.1 70B.

F Additional Qualitative Results

We include qualitative results in Fig. 7. The figure illustrates the action sequences produced by SCENEQUERYPLANNER and the REACT baseline when executing the instruction: “To prevent it from getting broken, place the wooden item from the desk into a box and put that box on the bed.” The intermediate decision processes for both planners are shown beneath their respective action sequences.

SCENEQUERYPLANNER correctly identifies the target objects through structured queries, whereas REACT relies solely on raw observations and consequently selects an incorrect object, leading to task failure. The corresponding full text trajectories are provided in List 7 and List 8.

We present additional qualitative results in Fig. 8. Both SCENEQUERYPLANNER and REACT initially fail to detect the target object. How-

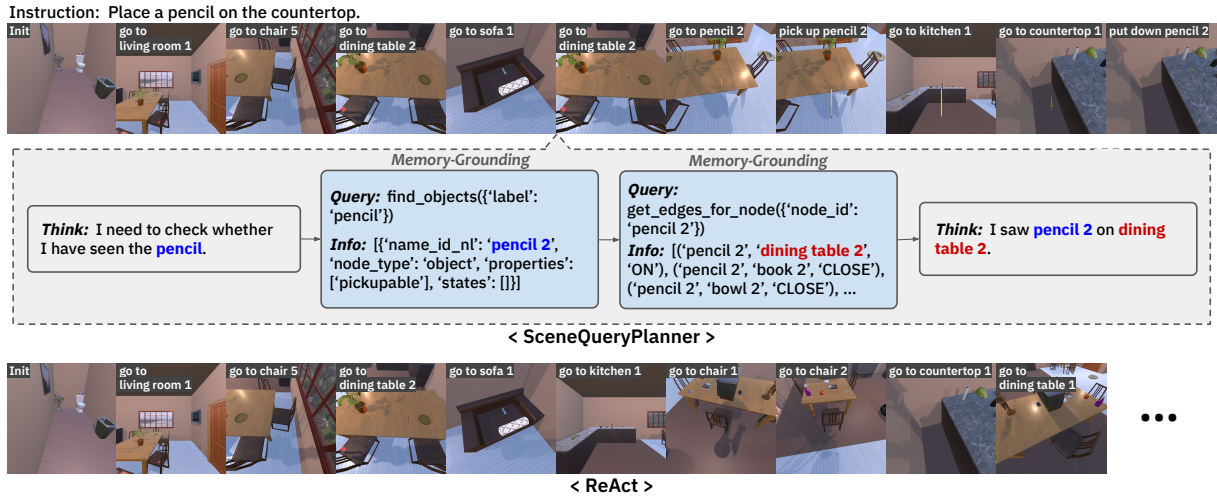


Figure 8: Comparison of SCENEQUERYPLANNER and REACT on the ATTPLAN-BASE task using LLaMA 3.1 70B. Although the pencil was located on dining table 2, both planners initially failed to recognize it and moved past the object. However, before leaving the room, SCENEQUERYPLANNER performs an error-correction step to verify whether the target object was missed, allowing it to recover from the mistake. In contrast, REACT fails to recover and repeatedly continues exploration, eventually leading to task failure.

ever, SCENEQUERYPLANNER revises its reasoning through active queries and recovers from the earlier mistake, ultimately completing the task successfully. In contrast, REACT fails to correct the initial error, and the mistake propagates through subsequent steps—eventually resulting in task failure. This example highlights the advantage of active information acquisition, which enables the agent to verify task-relevant facts via memory-grounding and recover from earlier errors during long-horizon planning.

Listing 7: Full Trajectory of SceneQueryPlanner

```
Your task is to: To prevent it from getting broken, place the wooden item from the desk into a box and put that box on the bed.
- You are in the house, and there are 3 rooms: bedroom (1), kitchen (1), living room (1). You are in the middle of a living room (1). Looking quickly around the room, you see TV stand (1), dining table (3), dog bed (1), shelf (1, 2, 3, 4, 5, 6), sofa (1).
- Think: To solve the task, I need to find and pick up the item made of wood from the desk, put it in box, pick up and place box on bed.
- OK.
- Think: First, I need to find and pick up the item made of wood from the desk. Now, I need to find desk.
- OK.
- Query: find_objects({'label': 'desk'})
- Info: [{'name_id_nl': 'desk 1', 'node_type': 'asset', 'properties': [], 'states': [], 'room': 'bedroom 1'}]
- Think: Desk 1 is in bedroom 1.
```

```
- OK.
- Act: go to bedroom 1
- You move to the bedroom (1). Looking quickly around the room, you see bed (1), chair (1, 2), desk (1), dining table (1), drawer (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12), dresser (1), garbage can (1), laundry hamper (1), painting (1, 2), television (1).
- Act: go to desk 1
- You arrive at the desk (1). You see alarm clock (1), bowl (1), chair (1), desk (1), house plant (1), mug (1), pencil (1).
- Think: Now, I need to find and pick up the item made of wood from the desk. First, I need to check what is on top of desk 1.
- OK.
- Query: get_edges_for_node({'node_id': 'desk 1', 'relation': 'ON'})
- Info: [('alarm clock 1', 'desk 1', 'ON'), ('bowl 1', 'desk 1', 'ON'), ('house plant 1', 'desk 1', 'ON'), ('mug 1', 'desk 1', 'ON'), ('pencil 1', 'desk 1', 'ON')]
- Think: Now, I need to check each object to find the item made of wood.
- OK.
- Query: read_node({'node_id': 'alarm clock 1'})
- Info: {'name_id_nl': 'alarm clock 1', 'salientMaterials': ['Metal', 'Plastic', 'Glass'], 'properties': ['pickupable'], 'states': []}
- Query: read_node({'node_id': 'bowl 1'})
- Info: {'name_id_nl': 'bowl 1', 'salientMaterials': ['Ceramic'], 'properties': ['canFillWithLiquid', 'breakable', 'pickupable'], 'states': []}
- Query: read_node({'node_id': 'house plant
```

```

1'))
- Info: {'name_id_nl': 'house plant 1',
'salientMaterials': ['Ceramic',
'Organic'], 'properties':
['canFillWithLiquid'], 'states': []}
- Query: read_node({'node_id': 'mug 1'})
- Info: {'name_id_nl': 'mug 1',
'salientMaterials': ['Ceramic'],
'properties': ['canFillWithLiquid',
'breakable', 'pickupable'], 'states':
[]}
- Query: read_node({'node_id': 'pencil 1'})
- Info: {'name_id_nl': 'pencil 1',
'salientMaterials': ['Wood'],
'properties': ['pickupable'], 'states':
[]}
- Think: On desk 1, the item made of wood
is pencil 1. Now, I need to pick it up.
- OK.
- Act: go to pencil 1
- You arrive at the pencil (1). You see
alarm clock (1), book (1), bowl (1),
chair (1), desk (1), drawer (1, 2, 5,
6, 7, 8, 9, 10, 11), dresser (1), house
plant (1), mug (1), pencil (1), statue
(1).
- Act: pick up pencil 1
- You pick up pencil (1). You hold pencil
(1).
- Think: Now, I need to find box and put
down pencil 1.
- OK.
- Query: find_objects({'label': 'box'})
- Info: [{'name_id_nl': 'box 2',
'node_type': 'asset', 'properties':
['openable', 'pickupable'], 'states':
[], 'room': 'bedroom 1'},
{'name_id_nl': 'box 1', 'node_type':
'asset', 'properties': ['openable',
'pickupable'], 'states': [], 'room':
'bedroom 1'}]
- Think: Box 1 is in bedroom 1.
- OK.
- Act: go to box 1
- You arrive at the box (1). The box (1) is
closed. You see basket ball (1), box
(1, 2), chair (2), dining table (1).
You hold pencil (1).
- Act: open box 1
- You open box (1). You see basket ball
(1), box (1, 2), chair (2), dining
table (1). You hold pencil (1).
- Act: put down pencil 1
- You put pencil (1) in box (1).
- Think: Now, I need to pick up box 1.
- OK.
- Act: pick up box 1
- You pick up box (1). You hold box (1).
- Think: Now, I need to find bed and put
down box 1.
- OK.
- Query: find_objects({'label': 'bed'})
- Info: [{'name_id_nl': 'bed 1',
'node_type': 'asset', 'properties': [],
'states': [], 'room': 'bedroom 1'}]
- Think: Bed 1 is in bedroom 1.
- OK.
- Act: go to bed 1
- You arrive at the bed (1). You see bed
(1), pencil (1), pillow (1, 2), tennis

```

```

racket (1). You hold box (1).
- Act: put down box 1
- You put box (1) on bed (1).
- Think: Now I complete the task.
- OK.
- Act: done

```

Listing 8: Full Trajectory of ReAct

```

Your task is to: To prevent it from getting
broken, place the wooden item from the
desk into a box and put that box on the
bed.
- You are in the house, and there are 3
rooms: bedroom (1), kitchen (1), living
room (1). You are in the middle of a
living room (1). Looking quickly around
the room, you see TV stand (1), dining
table (3), dog bed (1), shelf (1, 2, 3,
4, 5, 6), sofa (1).
- Think: To solve the task, I need to find
and pick up the item made of wood from
the desk, put it in box, pick up and
place box on bed.
- OK.
- Think: First, I need to find desk and
find the item made of wood from the
desk. The desk is more likely to appear
in bedroom 1.
- OK.
- Act: go to bedroom 1
- You move to the bedroom (1). Looking
quickly around the room, you see bed
(1), chair (1, 2), desk (1), dining
table (1), drawer (1, 2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 12), dresser (1), garbage
can (1), laundry hamper (1), painting
(1, 2), television (1).
- Act: go to desk 1
- You arrive at the desk (1). You see alarm
clock (1), bowl (1), chair (1), desk
(1), house plant (1), mug (1), pencil
(1).
- Think: On desk 1, the item made of wood
is bowl 1. Now, I need to pick it up.
- OK.
- Act: go to bowl 1
- You arrive at the bowl (1). You see alarm
clock (1), book (1), bowl (1), chair
(1), desk (1), drawer (1, 2, 5, 6, 7,
8, 9, 10, 11), dresser (1), house plant
(1), mug (1), pencil (1), statue (1).
- Act: pick up bowl 1
- You pick up bowl (1). You hold bowl (1).
- Think: Now, I need to find box and put
down bowl 1. The box is more likely to
appear in bedroom 1.
- Act: go to box 1
- You arrive at the box (1). The box (1) is
closed. You see basket ball (1), box
(1, 2), chair (2), dining table (1).
You hold bowl (1).
- Act: open box 1
- You open box (1). You see basket ball
(1), box (1, 2), chair (2), dining
table (1). You hold bowl (1).
- Act: put down bowl 1
- You put bowl (1) in box (1).
- Think: Now, I need to pick up box 1.

```

```

- OK.
- Act: pick up box 1
- You pick up box (1). You hold box (1).
- Think: Now, I need to find bed and put
  down box 1.
- OK.
- Act: go to bed 1
- You arrive at the bed (1). You see bed
  (1), pencil (1), pillow (1, 2), tennis
  racket (1). You hold box (1).
- Act: put down box 1
- You put box (1) on bed (1).
- Think: Now I complete the task.
- OK.
- Act: done

```

G 3DSG Update Procedure

At each time step t , the perception module produces a partial scene graph o_{t+1}^{graph} from the current observation. This partial graph is merged into the global belief state $\mathcal{G}_t$ via the update operator $\mathcal{G}_{t+1} = \text{UPDATE}(\mathcal{G}_t, o_{t+1}^{\text{graph}})$, which follows four rules applied in order:

Node update. Every node present in o_{t+1}^{graph} updates its corresponding entry in $\mathcal{G}_t$ with the latest observed attributes. New nodes not yet in $\mathcal{G}_t$ are inserted.

Active entity edges. For active entities—the agent and any object currently being grasped—all previously connected edges in $\mathcal{G}_t$ are removed and replaced entirely with the edges from o_{t+1}^{graph} . This prevents stale spatial relations from persisting as the agent moves through the environment.

Observed pair edges. For any two nodes that are both present in o_{t+1}^{graph} , all existing edges between them in $\mathcal{G}_t$ are deleted and replaced with the newly observed edges. This ensures that local spatial relationships within the current field of view are always up to date.

Maintenance. All nodes and edges in $\mathcal{G}_t$ that do not fall under the above conditions are preserved unchanged.

This incremental strategy ensures that the update cost scales linearly with the size of each partial observation rather than the full graph. Furthermore, it avoids serializing the complete graph into the LLM context—a practice that becomes intractable as environments grow—and instead enables selective retrieval via structured queries.

H Perception Noise Injection Details

We design five noise types that target complementary aspects of the perceived partial scene graph o_{t+1}^{graph} , simulating realistic perception failures in

Dataset	REACT		SQP	
	TSR	SSR	TSR	SSR
WAH-NL	44.00	58.75	47.00	59.78
ATTPLAN-BASE	65.00	71.73	68.40	74.72
ATTPLAN-ATTR	54.00	58.85	71.20	76.51

Table 8: Performance with GPT-4o as the backbone. SQP denotes SCENEQUERYPLANNER.

embodied environments: (1) **Node omission.** A node is removed from o_{t+1}^{graph} , simulating missed detections. (2) **Category confusion.** An object node’s semantic label is replaced with a visually similar category (*e.g.*, *mug* $\rightarrow$ *cup*, *bowl* $\rightarrow$ *plate*), simulating misclassification. (3) **Attribute flip.** A binary attribute of a node is inverted (*e.g.*, CLEAN $\rightarrow$ DIRTY, OPEN $\rightarrow$ CLOSED), simulating state estimation errors. (4) **Edge omission.** A spatial edge is removed, simulating failures in spatial relation detection. (5) **Edge relation swap.** An edge’s relation type is replaced with another (*e.g.*, ON $\rightarrow$ INSIDE), simulating spatial reasoning errors.

All five types are applied simultaneously at each step. For each eligible element (node, attribute, or edge), each noise type is triggered independently with a shared probability p . Corrupted partial graphs affect both the textual observation (derived from the graph) and the global 3DSG memory (via the update operator). REACT, which lacks graph memory, is affected only through the textual observation and is thus subject to noise types (1) and (2) only.

I GPT-4o Evaluation

Table 8 reports REACT and SCENEQUERYPLANNER with GPT-4o (Hurst et al., 2024) as the backbone, using the same in-context example pool and decision budget as in Sec. 5.1. Because GPT-4o does not expose token log-probabilities, we cannot apply the likelihood-based skill selection used for open-weight models. Instead, the model generates an action in free-form text, which is mapped to the closest executable skill by Sentence-BERT similarity (Reimers and Gurevych, 2019). This mapping occasionally introduces errors such as repetitive action loops, introducing an additional source of inference noise for both planners.