ONTOPLAN: An Ontology-Grounded Scene Representation and Agentic Framework for Scalable Robot Task Planning

Hyeongwoo Nam Woongje Cho* Juwon Kim* Jongeun Choi†
School of Mechanical Engineering, Yonsei University, Seoul, Republic of Korea
{namh_woo, woongje725, 2021145152, jongeunchoi}@yonsei.ac.kr

Abstract

Large language model (LLM)-based robot task planning is promising for open-ended instruction following, but degrades on long-horizon tasks in large environments. When spatial information is conveyed to the LLM through text, the model can fail to capture spatial context, and token cost grows with environment size. Generating action sequences directly with an LLM also makes it difficult to satisfy the current world state and action preconditions. We address this with an ontology-grounded scene representation that aligns objects, spaces, relations, and states in a shared symbolic vocabulary for spatial reasoning and task planning, and with **ONTOPLAN**, an agentic framework that interprets instructions, selectively retrieves task-relevant information, formalizes goals and constraints, and produces executable plans. Across 150 general tasks spanning five indoor environments and three scene scales, ONTOPLAN achieves 0.89 average task success, compared with 0.27 for the strongest baseline, while using 18.1k total tokens per task on average, about 5.6× fewer than the most efficient baseline. These advantages persist as scene scale increases, whereas prior methods degrade more sharply in success and remain far more costly in tokens. ONTOPLAN also responds appropriately to ambiguous or infeasible instructions by asking follow-up questions or reporting insufficient information rather than committing to invalid plans. Project page: https://namhyeongwoo.github.io/OntoPlan.

1 Introduction

Large language models (LLMs) are increasingly used as flexible interfaces for open-ended problem solving [1, 2, 3]. Rather than requiring rigid formal specifications, they let users express intent in natural language and use commonsense knowledge to organize underspecified requests into subgoals or solution steps [4, 5, 6, 7]. This makes them well suited to robot task planning, where instructions are open-ended and key details are often implicit. Robots must infer the intended task together with the relevant objects, spaces, and constraints.

Robot task planning, however, requires more than language understanding. A valid solution must ground task-relevant objects, spaces, and relations in large scenes and generate long-horizon action sequences that remain executable under the current world state. This exposes two limitations of LLM-only planning. First, flattening structured spatial representations such as 3D scene graphs into text prompts can cause spatial reasoning errors, including hallucinated or omitted relations, while prompt cost grows with scene size [8, 9]. Second, directly generated action sequences are unreliable under grounded execution, often violating action preconditions, safety or feasibility constraints, or

*Equal contribution.
†Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

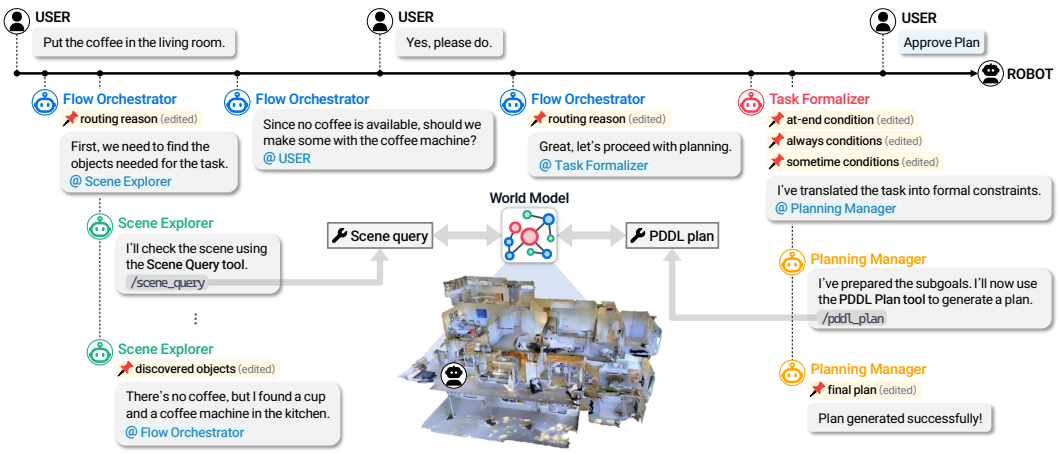


Figure 1: **Overview of OntoPlan.** From a user instruction, the Flow Orchestrator routes scene grounding, clarification, task formalization, and planning across specialized agents, while the *Scene query* and *PDDL plan* tools operate over a shared ontology-grounded world model.

the intended goal [10, 11]. Alternatively, LLMs can formulate planner-facing PDDL problems for symbolic planners, but this step remains vulnerable to errors such as missing initial facts, misused predicates, and incomplete goals or constraints [12, 13]. The central challenge is to maintain consistency across instruction interpretation, scene grounding, and executable planning.

We address this challenge with **OntoPlan**, which separates the roles of LLMs and structured tools. LLM agents handle high-level instruction interpretation, clarification, and goal and constraint formalization, while structured tools handle selective scene access and executable long-horizon planning. To make this separation reliable, OntoPlan maintains a shared symbolic state in an ontology-grounded world model aligned directly with a PDDL domain [14]. The *Scene query* tool retrieves task-relevant objects, spaces, and relations from this shared state, while the *PDDL plan* tool generates executable plans from the same world state and explicit constraints. Figure 1 illustrates how the shared world model connects scene grounding, task formalization, planning, and world-state updates.

This paper makes two contributions. First, we propose a PDDL-aligned ontology-grounded scene representation that supports shared scene grounding, direct world-state extraction, and consistent symbolic updates. Second, we develop a tool-using planning framework that combines the *Scene query* tool for selective scene access with the *PDDL plan* tool for executable long-horizon planning. We evaluate OntoPlan across five indoor environments and three scene scales, and further analyze its behavior on ambiguous and infeasible instructions. OntoPlan improves task success and token efficiency over LLM-based baselines and remains more robust as scenes grow or when instructions are ambiguous or infeasible.

2 Related Work

Structured Scene Grounding. Structured scene representations encode objects, spaces, and relations, providing a more reliable substrate for spatial reasoning than free-form scene descriptions [15, 16, 17, 18, 19, 20]. SayPlan retrieves task-relevant context from hierarchical 3D scene graphs, passes it to an LLM as text, and replans using feedback from a scene-graph simulator [21]. While such methods demonstrate the value of structured grounding, they still route graph structure through natural-language context. Flattening graph-structured information into text can obscure relational dependencies [8, 22] and increases prompt length as scenes grow, which can degrade the model's use of long contexts [23, 24]. This motivates keeping scene knowledge queryable in symbolic form and retrieving only task-relevant evidence [25, 26].

LLM-Assisted Symbolic Planning. Direct LLM planning remains unreliable on formal planning tasks, especially for long-horizon validity and reasoning about state change [10, 27, 28, 29]. Hybrid approaches instead use LLMs to interpret instructions or construct planner-facing specifications,

then delegate planning to external symbolic planners [30, 31]. Prior work has shown that LLMs can translate natural-language goals into formal planning goals [4], construct or correct PDDL models [32, 33, 34], and convert natural-language problems into PDDL [12, 35, 36]. DELTA combines scene-graph pruning with autoregressive subgoal decomposition for embodied long-horizon planning [13]. These approaches reduce reliance on direct plan generation, but remain vulnerable to the formalization errors noted in Section 1. NL-PDDL extends PDDL into a natural-language representation for lifted regression planning with commonsense entailment in partially observed environments [37], whereas ONTOPLAN assumes full observability with a typed vocabulary shared by the world model and PDDL domain.

Interactive Planning under Ambiguity. Interactive planning work studies how systems should respond to underspecified instructions or when execution feedback reveals failures [38, 39]. Inner Monologue incorporates scene descriptions, success signals, and human feedback into closed-loop planning [40]. DEPS revises plans using execution descriptions and self-explanations while selecting among subgoals [41]. Introspective Planning explicitly reasons about uncertainty under ambiguous instructions and asks follow-up questions when needed [42, 43]. These studies motivate planning loops that support clarification and feedback-driven revision rather than single-pass plan generation.

Positioning. Taken together, prior work spans perceptually grounded construction and querying of spatial and spatio-temporal memories [17, 20, 44] as well as planning over structured scene representations, but does not fully integrate selective access to large structured scenes with executable planning over the same symbolic state. Such memories could serve as upstream inputs after mapping their contents to the ontology–PDDL vocabulary, while ONTOPLAN uses the resulting symbolic state for scene grounding, task formalization, planner-state extraction, primitive-action synthesis, and symbolic state update. This architecture instantiates the LLM-modulo framework [30] and follows the broader tool-use paradigm for language models [45], supporting consistency across high-level reasoning and grounded execution.

3 ONTOPLAN

ONTOPLAN assumes a fully observable scene represented by symbolic instances and state facts, together with a predefined primitive action set. Its planning loop is organized around a shared ontology-grounded world model aligned with the PDDL domain and two tool interfaces: the *Scene query* tool for scene grounding, which links instruction phrases to scene entities and relations, and the *PDDL plan* tool for generating executable plans. LLM agents coordinate these tools through shared memory while interpreting instructions, requesting clarification, and revising intermediate results.

3.1 World Model

The world model is the shared symbolic representation for scene understanding, scene grounding, and planning. As shown in Figure 2, it consists of three layers: an ontology layer declaring entity types and relation semantics, a static layer storing persistent topology and connectivity, and a dynamic layer storing runtime object instances, states, and spatial relations. Across environments, this fixed schema is instantiated with environment-specific symbolic facts, which in our experiments come from the benchmark scene annotations: topology and connectivity populate the static layer, while object instances, states, and spatial relations populate the dynamic layer. This separates the reusable schema and persistent environment structure from task-time state.

A key design choice is to mirror the ontology directly in the PDDL domain definition: ontology classes map one-to-one to PDDL object types, ontology properties map one-to-one to predicates with the same names, and their associated logical constraints are aligned across the two representations. As a result, scene grounding, task formalization, and planning operate over a shared ontology–PDDL vocabulary, avoiding an intermediate schema-translation stage and keeping grounded context consistent with planner state.

Access to the world model is provided through an ask/tell interface. The *ask* operation executes read-only SPARQL queries for scene inspection and current-state extraction, whereas *tell* updates the dynamic layer by inserting and deleting RDF triples for symbolic observations and action effects. In our implementation, the fully observed scene state is stored as asserted triples. When these triples are loaded or updated, GraphDB applies the declared OWL 2 RL semantics through forward chaining

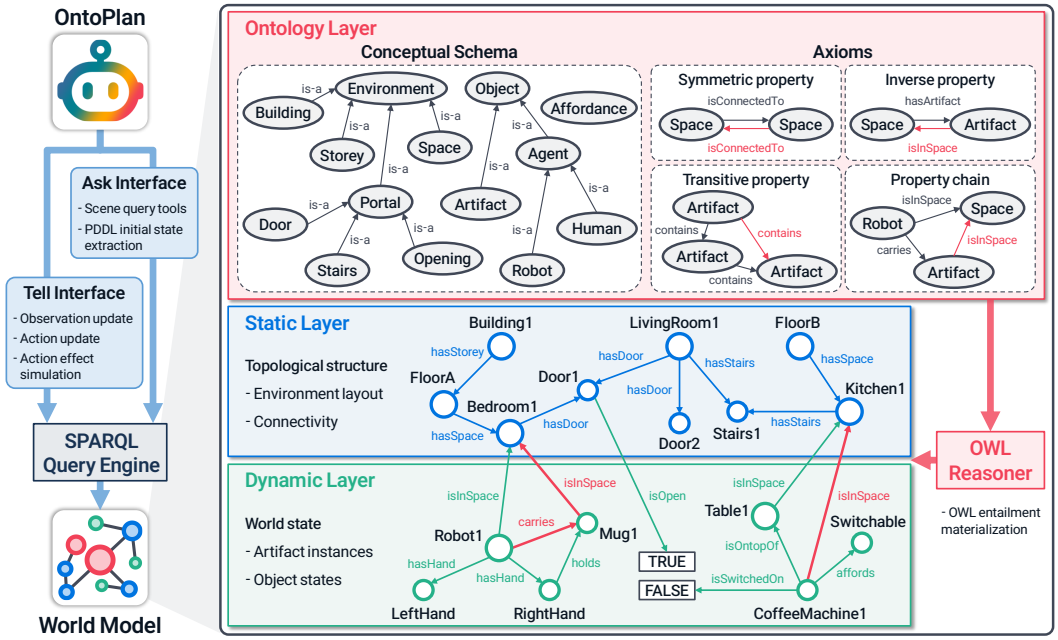


Figure 2: **World Model.** Ontology-grounded representation with ontology, static, and dynamic layers; *ask* supports scene inspection and planner-state extraction, and *tell* updates dynamic facts from symbolic observations and action effects. Red edges denote facts inferred by the OWL reasoner.

and materializes the resulting entailments. The *ask* operation then queries both asserted facts and materialized inferences. For example, `carries` is defined by the property chain `hasHand`∘`holds`, and an artifact's space is derived through `isOntopOf`, `isInsideOf`, and `isCarriedBy`, so an object carried by the robot is inferred to share the robot's location; in the Marstons Medium scene, materialization expands 2,908 asserted triples to 19,394. The tool interfaces described next are built on this ask/tell mechanism. Appendix A provides further update and inference examples.

3.2 System Architecture

As shown in Figure 3, ONTOPLAN is organized around a shared scene representation, two tool interfaces, and shared memory. The *Scene query* tool grounds task-relevant scene evidence, while the *PDDL plan* tool generates executable plans over the current world state. Shared memory stores the current intent interpretation, grounding evidence, unresolved ambiguities, formalized goals and constraints, and planner diagnostics, allowing agents to revise intermediate results as planning proceeds.

3.2.1 Tool Interfaces

ONTOPLAN exposes the world model and planner through two ask/tell-based tool interfaces.

Scene query. The *Scene query* tool inspects graph-structured knowledge in the world model through two functions. `semantic_search` combines embedding-based retrieval with SPARQL filtering to identify task-relevant objects and spaces, while `find_path` computes shortest paths over the scene connectivity graph using SPARQL. During scene grounding, the agent issues task-dependent queries, records the returned evidence in shared memory, and refines follow-up queries as needed. When restrictive filters return no match, `semantic_search` can broaden the query scope and return hint fields with similar categories. Scene grounding thus proceeds as iterative evidence gathering over the symbolic scene representation, rather than by serializing the full scene graph into the prompt.

PDDL plan. The *PDDL plan* tool synthesizes or repairs long-horizon executable plans from a formalized task specification. It extracts the current world state from the world model, compiles trajectory-level `always` conditions into the planning model, and solves ordered subgoals sequentially. Each planning problem is built with SPARQL from the artifacts named in the subgoal, the containers,

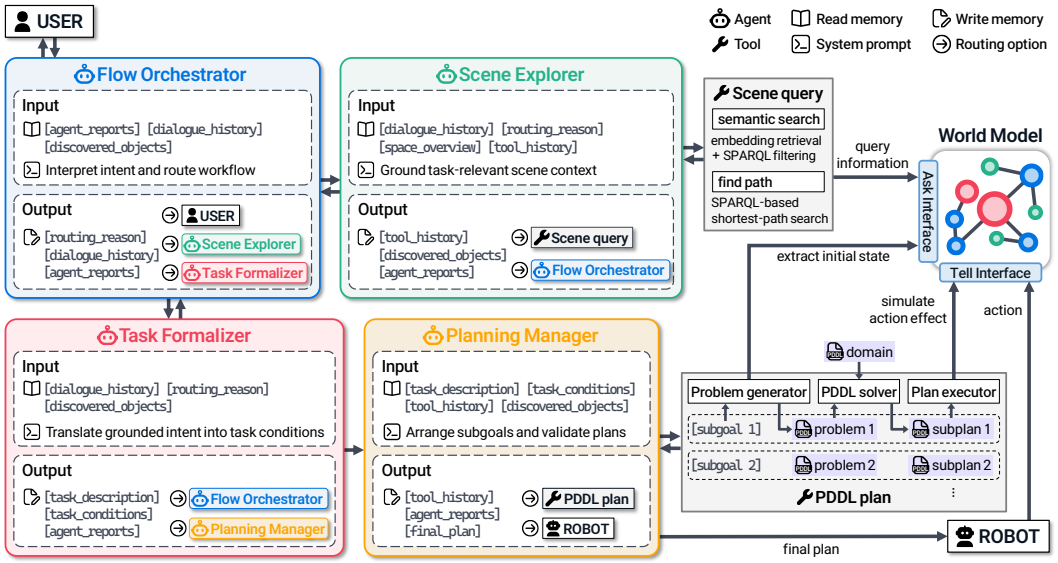


Figure 3: **System Architecture of OntoPlan.** Flow Orchestrator, Scene Explorer, Task Formalizer, and Planning Manager coordinate through shared memory, while the *Scene query* and *PDDL plan* tools access the shared world model for scene grounding, subgoal planning, and revision. Each agent box lists the memory fields the agent reads (Input) and writes (Output), its role in the system prompt, and its routing options.

surfaces, and contents linked to them, the full navigation topology, and the robot state, so unrelated objects do not enlarge it. Because Fast Downward [46], the planner in our implementation, does not support PDDL 3 trajectory constraints, each `always` condition is compiled into the domain as a violation flag that any action falsifying the condition sets, and every problem requires the flag to remain false. After each successful subgoal, the tool simulates action effects, writes the resulting updates back to the world model, and uses the updated state for the next subgoal, keeping the parts of the final goal already achieved as goals so that later subgoals do not undo them. The tool returns either a plan or failure diagnostics, so the agent workflow does not need to construct a full PDDL problem directly, and action-level validity and state transitions are handled by Fast Downward rather than by the LLM.

3.2.2 Agent Workflow

On top of these tools, OntoPlan factorizes the planning loop into four roles with distinct responsibilities: workflow control, scene grounding, task formalization, and executable planning/revision. This decomposition narrows each agent's reasoning scope while preserving a shared symbolic task context. Detailed prompts are provided in Appendix B.

Flow Orchestrator. The Flow Orchestrator maintains the current intent interpretation from the instruction, dialogue history, and shared memory. It routes the workflow by deciding whether to ask the user for clarification, continue scene grounding, or proceed to task formalization.

Scene Explorer. The Scene Explorer uses the *Scene query* tool to retrieve task-relevant objects, spaces, relations, and states from the world model [31]. It records the evidence and candidate entities in shared memory, then returns control once the task is sufficiently grounded or when missing or ambiguous facts require further routing.

Task Formalizer. The Task Formalizer converts the interpreted intent and grounded scene context into planner-facing final-state goals and trajectory constraints. If the grounding evidence is sufficient, it writes the formalized conditions to shared memory and hands them to the Planning Manager; otherwise, it requests further grounding.

Planning Manager. The Planning Manager organizes the formalized goals and constraints into ordered subgoals, invokes the *PDDL plan* tool on the current world state, and interprets the resulting

plan or failure diagnostics. When planning fails, it uses this feedback to revise the subgoal structure and trigger replanning.

4 Experimental Setup

We evaluate ONTOPLAN on long-horizon robot task planning in large-scale indoor environments. Our evaluation uses five indoor environments, three scene scales, 150 general instructions, and two diagnostic subsets (ambiguous and infeasible). We compare against SayPlan and DELTA under shared LLM and planner settings using planning reliability, task success, and token-usage metrics.

4.1 Datasets

Environments. The environment dataset consists of five indoor spaces from the 3D Scene Graph dataset [16]: Klickitat, Lakeville, Lindenwood, Marstons, and Muleshoe. We deterministically map the original annotations into a fixed ontology schema to form the *Small* split, manually augment the resulting scenes to form the *Medium* split, and derive the *Large* split from *Medium* through constrained random augmentation to triple the object count. All scales share the same ontology and PDDL schema. Table 1 reports the structural statistics and object counts, and Appendix C visualizes the scene graphs across scales.

Table 1: **Evaluation Environment Statistics.** Floors, rooms, and object counts for the five environments at Small, Medium, and Large scales.

Name	Floors	Rooms	Objects		
			Small	Medium	Large
Klickitat	3	28	84	385	1155
Lakeville	2	19	83	387	1161
Lindenwood	2	36	59	318	954
Marstons	4	31	50	388	1164
Muleshoe	2	24	121	423	1269

Task Instructions. For each environment and scale, we manually wrote 10 general task instructions, yielding 150 general tasks in total. We also constructed two diagnostic subsets with 10 tasks each: *Ambiguous* and *Infeasible*. The general tasks form the primary evaluation set, while the diagnostic subsets are used to analyze behavior under underspecified or unsatisfiable requests rather than to support standalone statistical claims. Representative examples are shown in Table 2, and additional examples are provided in Appendix D. Each task is paired with evaluation checks used for scoring: `at_end`, `always`, and `sometime`, which specify the final goal, trajectory-wide constraints, and intermediate states that must be achieved at least once, and, for six tasks, `ordering`, which requires one state to first hold no later than another.

Table 2: **Representative Task Instructions.** Abridged examples from the three subsets, with gray annotations summarizing their evaluation context.

Type	Examples
General	I want to ventilate the house. Open all closed doors, except the home office door. → (always) keep home office door closed. → (at_end) all other doors are open.
	Wash Sam's T-shirt from the bed on Floor A in the washer, then put it back on the bed. → (sometime) T-shirt in the running washer. → (at_end) T-shirt is back on the bed.
Ambiguous	Bring the book I was reading yesterday from the living room to my room. → (ambiguity) book and room unspecified.
Infeasible	I want to skateboard again after a long time. Find the skateboard in the living room on Floor B and bring it to the lobby. → (infeasibility) no skateboard in the environment.

4.2 Evaluation Metrics

The evaluation checks are used only for scoring; neither ONTOPLAN nor the baselines receive them as input. *Plan Returned* measures whether the system returns a symbolic plan for the task. Each returned action sequence is then symbolically executed from the current state until either all actions are applied or the first precondition failure occurs; Appendix E provides the full symbolic evaluation procedure. *Plan Executable* measures whether the full sequence completes without failure. *Goal Reached* measures whether the reached state satisfies the `at_end` condition; because this state may result from an executable prefix, Goal Reached can exceed Plan Executable. *Task Success* requires

Table 3: **General-Task Performance by Environment.** Results are averaged over the three scene scales for each environment. All four performance metrics are rates, and token metrics are reported in $\times 10^3$. Bold marks the best value in each environment.

Environment	Method	Plan Returned	Plan Executable	Goal Reached	Task Success	Input Tokens per Call	Total Tokens per Task
Klickitat	SayPlan	0.67	0.47	0.33	0.27	30.9	221.7
	DELTA	0.63	0.23	0.23	0.20	23.9	101.3
	ONTOPLAN	**1.00**	**1.00**	**0.93**	**0.90**	**2.6**	**17.3**
Lakeville	SayPlan	0.57	0.40	0.33	0.33	33.3	205.5
	DELTA	0.47	0.23	0.17	0.17	23.4	98.8
	ONTOPLAN	**1.00**	**1.00**	**0.97**	**0.97**	**2.6**	**18.8**
Lindenwood	SayPlan	0.63	0.47	0.40	0.40	34.5	206.6
	DELTA	0.37	0.17	0.23	0.17	23.4	98.9
	ONTOPLAN	**1.00**	**1.00**	**0.87**	**0.87**	**2.6**	**17.5**
Marstons	SayPlan	0.73	0.43	0.30	0.27	37.1	255.6
	DELTA	0.27	0.17	0.17	0.10	23.9	101.3
	ONTOPLAN	**1.00**	**1.00**	**0.77**	**0.77**	**2.6**	**18.7**
Muleshoe	SayPlan	0.70	0.17	0.20	0.10	31.9	190.4
	DELTA	0.47	0.03	0.03	0.03	24.2	102.7
	ONTOPLAN	**0.97**	**0.97**	**0.93**	**0.93**	**2.6**	**18.1**

full execution and satisfaction of all applicable checks, including `at_end`, `always`, `sometime`, and `ordering`.

Efficiency is measured using *Total Tokens per Task*, *Input Tokens per Call*, and, in the ablation, *Peak Input Tokens per Call*. The first sums input and output tokens over all LLM calls for a task.[3] The second averages input tokens over those calls. The third is the largest single-call input per task.

4.3 Implementation Details

We compare ONTOPLAN with SayPlan and DELTA under matched task instances, initial states, LLM settings, and success checks. We use the released DELTA codebase and implement SayPlan following its original hierarchical scene-graph retrieval pipeline. All methods use GPT-4o with temperature 0 [47], share the same 16 primitive actions, and are evaluated with the same action preconditions, effects, and checks. DELTA and ONTOPLAN start from the same PDDL domain and use the same Fast Downward configuration [46] (lazy weighted A* with the FF heuristic, $w = 2$), and SayPlan and ONTOPLAN may use up to 20 scene-exploration LLM calls and 5 plan-generation calls. Appendix E gives further details.

The main quantitative comparison uses a non-interactive protocol. Because SayPlan and DELTA do not support user interaction during planning, ONTOPLAN also runs with a fixed automatic reply to clarification requests in this setting: "*Do not ask follow-up questions. Make reasonable assumptions and continue planning.*" For the two diagnostic subsets, we report additional analyses: for the ambiguous subset, a user-interactive setting in which a human answers ONTOPLAN's clarification questions; and for the infeasible subset, the initial response types to unsatisfiable requests.

5 Results

We organize the results around five questions: general-task success and token efficiency (Table 3), behavior on ambiguous and infeasible instructions (Tables 4 and 5), scaling with scene size (Figure 4), the contribution of the two tools (Table 6), and the full workflow in qualitative multi-turn examples (Figure 5).

[3]This includes calls in runs that terminated after exceeding the model's context limit, which occurs only for SayPlan.

Table 4: **Performance on Ambiguous Instructions.** Results on 10 ambiguous tasks, reported as successful cases. Auto replies use the fixed reply of Section 4.3; user replies answer the clarification questions directly.

Method	Plan Returned	Plan Executable	Goal Reached†
SayPlan	5/10	4/10	0/10
DELTA	5/10	3/10	0/10
ONTOPLAN (auto replies)	8/10	8/10	1/10
ONTOPLAN (user replies)	**10/10**	**10/10**	**8/10**

† Only `at_end` is checked; Task Success is omitted.

Table 5: **Responses to Infeasible Instructions.** Results on 10 infeasible tasks, reported as response counts by category.

Method	Clarification Request†	Misdirected Plan	Inexecutable Plan	Planning Failure
SayPlan	—	3/10	0/10	7/10
DELTA	—	1/10	1/10	8/10
ONTOPLAN	**9/10**	1/10	0/10	0/10

† Not supported for SayPlan or DELTA.

5.1 Performance on General Tasks

Table 3 shows that ONTOPLAN consistently outperforms both baselines across all five environments, reaching 0.89 average task success versus 0.27 for SayPlan and 0.13 for DELTA while using 18.1k total tokens per task versus 216.0k and 100.6k, respectively. Its smaller gap between Plan Returned and Plan Executable indicates that its returned plans more often remain executable under the evaluated world state, while the lower token cost reflects selective scene querying. Appendix F provides paired 95% bootstrap confidence intervals, stratified by scene scale, for the four binary performance metrics in Table 3. All 149 plans returned by ONTOPLAN execute without precondition failures; its remaining failures arise before planning, mostly from grounding a referring expression to the wrong object or place (Appendix G). Baseline plans go unreturned mainly because of incomplete specifications or invalid search commands, not timeouts (Appendix H).

5.2 Handling Ambiguous and Infeasible Instructions

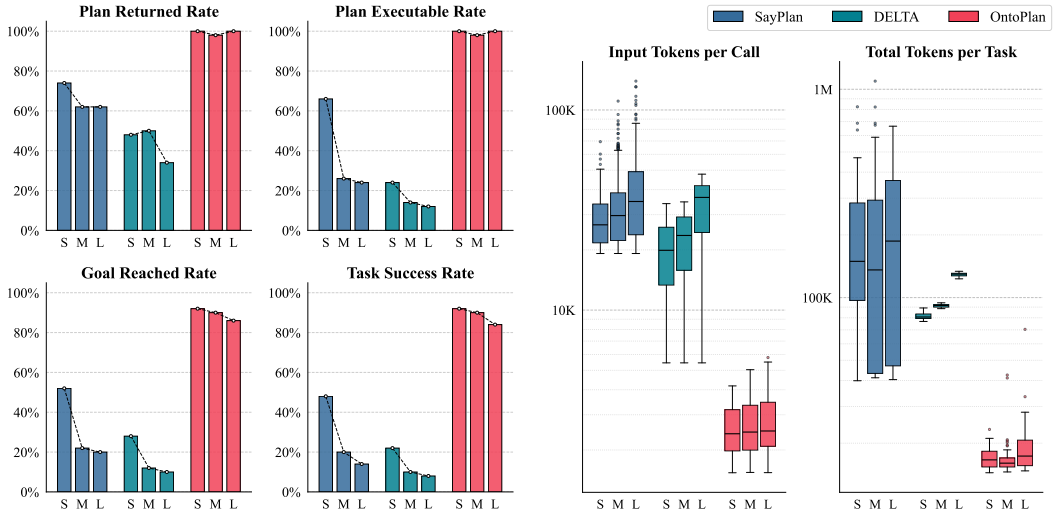
The ambiguous and infeasible subsets reveal behavioral differences not captured by aggregate success rates. Table 4 shows that baselines recover poorly from underspecified requests, whereas ONTOPLAN uses clarification before planning; with user answers instead of the fixed automatic reply from Section 4.3, it reaches the goal in 8/10 tasks instead of 1/10. For infeasible tasks, Table 5 groups responses into four categories: *Clarification Request*, when the system identifies missing, conflicting, or infeasible grounding evidence and asks for clarification or an alternative; *Misdirected Plan*, when the plan targets the wrong object or goal; *Inexecutable Plan*, when symbolic execution fails; and *Planning Failure*, when no plan is returned. Because SayPlan and DELTA cannot request clarification in our protocol, they mostly fall into less desirable categories, especially planning failure. In contrast, ONTOPLAN most often asks for clarification rather than committing to an invalid plan. Full per-instruction results are provided in Appendices I and J.

5.3 Scalability Analysis

Figure 4(a)–(b) shows that ONTOPLAN maintains the highest success rates across all four metrics as scene complexity increases from Small to Medium to Large, while its input and total token usage remain comparatively flat across scales. By contrast, SayPlan shows larger variance and heavier token tails, and DELTA remains substantially more token-intensive than ONTOPLAN at larger scales.

5.4 Ablation Study

Table 6 compares the full system with two interface-level variants under matched task and evaluation settings. Without the *Scene query* tool, the full scene is serialized into the prompt, so the LLM must identify task-relevant information from the entire scene description. The Large-to-Small ratio consequently increases from 1.15× to 4.02× for *Total Tokens per Task* and from 1.06× to 8.13× for *Peak Input Tokens per Call*, indicating that selective retrieval is central to the observed token scalability, while task success remains statistically indistinguishable from that of the full system (Appendix F). In the w/o *PDDL plan* variant, the *Scene query* tool is retained, but the integrated planning path—ordered-subgoal planning, Fast Downward search, symbolic state-transition handling,



(a) Bar plots of success metrics by scene scale.

(b) Box plots of input tokens per call and total tokens per task by scene scale (log scale).

Figure 4: **Scalability Across Scene Scales.** ONTOPLAN degrades more gracefully than the baselines while remaining more token-efficient as scene complexity increases. Each scale comprises 50 tasks, and dashed lines connect a method's rates across scales.

Table 6: **Ablation of *Scene query* and *PDDL plan* Across Scene Scales.** The w/o *Scene query* and w/o *PDDL plan* variants replace selective retrieval with full-scene serialization and the integrated planning path with direct LLM action-sequence generation, respectively. Avg. denotes the mean across scales; token metrics are reported in $\times 10^3$, and L/S denotes the Large-to-Small ratio.

Setting	Task Success				Total Tokens per Task				Peak Input Tokens per Call			
	S	M	L	Avg.	S	M	L	L/S	S	M	L	L/S
Full system	**0.92**	**0.90**	**0.84**	**0.89**	**17.1**	**17.5**	**19.7**	1.15×	**3.6**	**3.7**	**3.9**	1.06×
w/o Scene query	0.86	**0.90**	**0.84**	0.87	18.1	32.5	72.7	4.02×	7.6	22.3	62.2	8.13×
w/o PDDL plan	0.32	0.26	0.36	0.31	23.6	25.0	26.9	**1.14×**	8.4	8.5	8.4	**1.01×**

and diagnostic-based replanning—is replaced by a single LLM call that receives the filtered scene context, the formalized task conditions, and the same PDDL domain, and generates the complete primitive-action sequence. Task success decreases from 0.89 to 0.31, reflecting the integrated planning path as a whole rather than Fast Downward search or ontology–PDDL alignment in isolation. Overall, the results associate selective scene access with scalability and the integrated planning path with task reliability.

5.5 Qualitative Analysis

Figure 5 illustrates how ONTOPLAN uses a shared world model across question answering, task grounding, executable planning, and follow-up execution. The system retrieves only task-relevant scene facts, translates them into planner-ready goals and constraints, and invokes symbolic planning over the current state. The trace highlights three capabilities not captured by aggregate metrics: path-based spatial question answering, grounding of implicit placement choices such as suitable support surfaces, and follow-up planning after action effects update the world model. Together, they show the two tools interacting over a shared symbolic state across user turns.

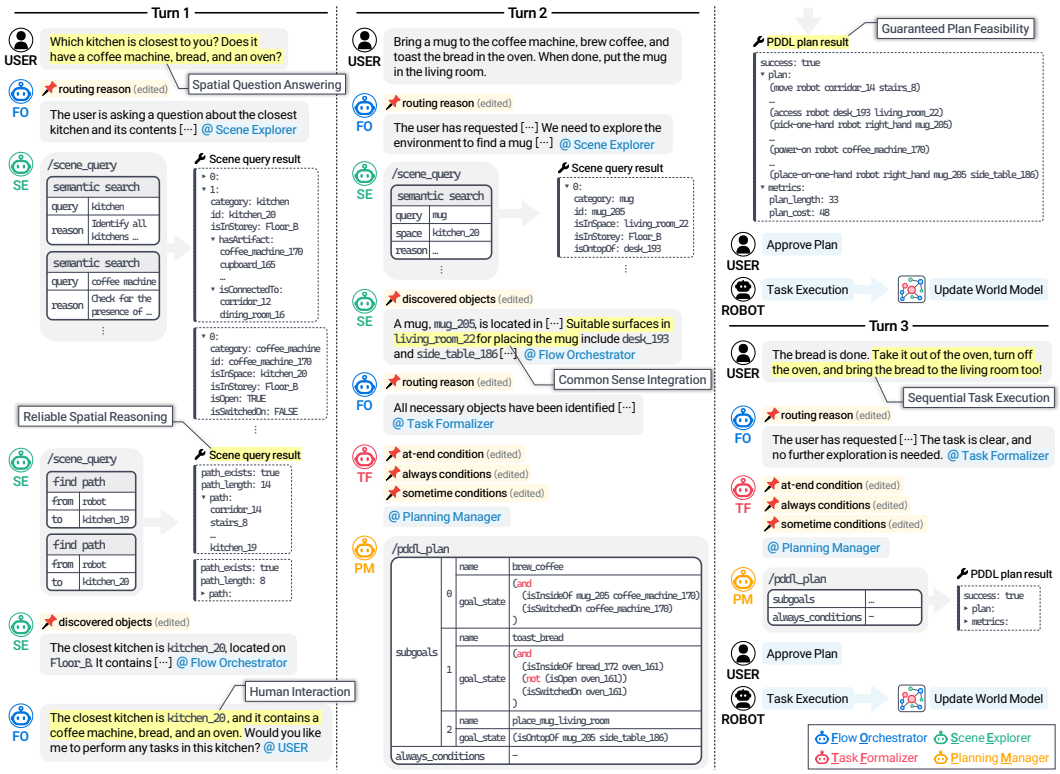


Figure 5: **Qualitative Examples of OntoPlan.** A trace of spatial question answering, common-sense task grounding, executable planning, and sequential execution with world model updates.

6 Conclusion

OntoPlan shows that scalable LLM-based robot task planning benefits from delegating structured scene access and executable plan synthesis to dedicated tools over a shared symbolic representation. Its ontology-grounded world model aligns directly with the PDDL domain, allowing scene grounding, task formalization, state extraction, and action-effect updates to use the same symbolic vocabulary. Across the evaluated environments, OntoPlan improves task success and token efficiency relative to representative baselines while better handling ambiguity and infeasibility.

The current study assumes a fully instantiated, fully observable symbolic world model and does not evaluate raw-sensor perception, learned scene reconstruction, or partial observability, so the results should be interpreted as evidence for planning quality rather than end-to-end real-world autonomy. Extending OntoPlan to partial observability will require perception and state-estimation modules [17, 20] that convert sensor observations into asserted symbolic updates. The ontology can enrich these asserted facts through declared entailments, but does not recover unobserved state by itself. The ontology–PDDL alignment also adds a modeling cost. Runtime state changes modify only dynamic facts, and new environments expressible under the existing schema and action set require only new static and instance facts, whereas genuinely new semantics or robot capabilities require coordinated extensions to both the ontology and the PDDL domain. Automatic cross-domain transfer, schema and action-model induction, and integration with vision-language-action models and real robot systems remain future work.

Acknowledgments and Disclosure of Funding

This work was supported by the National Research Foundation of Korea (NRF) grants funded by the Korea government (MSIT) (Nos. RS-2024-00344732 and RS-2026-25505230) and by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2026-25524880). The authors declare no competing interests.

References

[1] Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Ziyue Li, Xingyu Zeng, and Rui Zhao. TPTU: Task planning and tool usage of large language model-based AI agents. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

[2] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pages 50528–50652, 2024.

[3] Liwei Jiang, Yuanjun Chai, Margaret Li, Mickel Liu, Raymond Fok, Nouha Dziri, Yulia Tsvetkov, Maarten Sap, and Yejin Choi. Artificial hivemind: The open-ended homogeneity of language models (and beyond). In *Advances in Neural Information Processing Systems*, volume 38, 2025.

[4] Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*, 2023.

[5] Yijun Yang, Tianyi Zhou, Kanxue Li, Dapeng Tao, Lusong Li, Li Shen, Xiaodong He, Jing Jiang, and Yuhui Shi. Embodied multi-modal agent trained by an LLM from a parallel TextWorld. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 26275–26285, 2024.

[6] Nathaniel Weir, Muhammad Khalifa, Linlu Qiu, Orion Weller, and Peter Clark. Learning to reason via program generation, emulation, and search. In *Advances in Neural Information Processing Systems*, volume 37, pages 36390–36413, 2024.

[7] Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. In *Advances in Neural Information Processing Systems*, volume 36, pages 31967–31987, 2023.

[8] Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. Can language models solve graph problems in natural language? In *Advances in Neural Information Processing Systems*, volume 36, pages 30840–30861, 2023.

[9] Xixi Wu, Yifei Shen, Caihua Shan, Kaitao Song, Siwei Wang, Bohang Zhang, Jiarui Feng, Hong Cheng, Wei Chen, Yun Xiong, and Dongsheng Li. Can graph learning improve planning in LLM-based agents? In *Advances in Neural Information Processing Systems*, volume 37, pages 5338–5383, 2024.

[10] Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. PlanBench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Advances in Neural Information Processing Systems*, volume 36, pages 38975–38987, 2023.

[11] Wenlong Huang, Fei Xia, Dhruv Shah, Danny Driess, Andy Zeng, Yao Lu, Pete Florence, Igor Mordatch, Sergey Levine, Karol Hausman, and Brian Ichter. Grounded decoding: Guiding text generation with grounded models for embodied agents. In *Advances in Neural Information Processing Systems*, volume 36, pages 59636–59661, 2023.

[12] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

[13] Yuchen Liu, Luigi Palmieri, Sebastian Koch, Ilche Georgievski, and Marco Aiello. DELTA: Decomposed efficient long-term robot task planning using large language models. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10995–11001, 2025.

[14] Drew McDermott, Malik Ghallab, Adele E. Howe, Craig A. Knoblock, Ashwin Ram, Manuela Veloso, Daniel S. Weld, and David E. Wilkins. PDDL—the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[15] Michael Beetz, Daniel Beßler, Andrei Haidu, Mihai Pomarlan, Asil Kaan Bozcuoğlu, and Georg Bartels. KnowRob 2.0 – a 2nd generation knowledge processing framework for cognition-enabled robotic agents. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519, 2018.

[16] Iro Armeni, Zhi-Yang He, JunYoung Gwak, Amir R. Zamir, Martin Fischer, Jitendra Malik, and Silvio Savarese. 3D scene graph: A structure for unified semantics, 3D space, and camera. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5664–5673, 2019.

[17] Nathan Hughes, Yun Chang, and Luca Carlone. Hydra: A real-time spatial perception system for 3D scene graph construction and optimization. In *Proceedings of Robotics: Science and Systems*, 2022.

[18] Nathan Hughes, Yun Chang, Siyi Hu, Rajat Talak, Rumaia Abdulhai, Jared Strader, and Luca Carlone. Foundations of spatial perception for robotics: Hierarchical representations and real-time systems. *The International Journal of Robotics Research*, 43(10):1457–1505, 2024.

[19] 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. ConceptGraphs: Open-vocabulary 3D scene graphs for perception and planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5021–5028, 2024.

[20] Nicolas Gorlo, Lukas Schmid, and Luca Carlone. Describe anything anywhere at any moment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 35002–35013, 2026.

[21] Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. SayPlan: 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, 2023.

[22] Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

[23] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the Middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

[24] Mosh Levy, Alon Jacoby, and Yoav Goldberg. Same task, more tokens: the impact of input length on the reasoning performance of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15339–15353, 2024.

[25] Christopher Agia, Krishna Murthy Jatavallabhula, Mohamed Khodeir, Ondrej Miksik, Vibhav Vineet, Mustafa Mukadam, Liam Paull, and Florian Shkurti. Taskography: Evaluating robot task planning over large 3D scene graphs. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 46–58, 2022.

[26] Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V. Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. G-Retriever: Retrieval-augmented generation for textual graph understanding and question answering. In *Advances in Neural Information Processing Systems*, volume 37, pages 132876–132907, 2024.

[27] Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models: A critical investigation. In *Advances in Neural Information Processing Systems*, volume 36, pages 75993–76005, 2023.

[28] Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. Chain of Thoughtlessness? An analysis of CoT in planning. In *Advances in Neural Information Processing Systems*, volume 37, pages 29106–29141, 2024.

[29] Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li Erran Li, Ruohan Zhang, Weiyu Liu, Percy Liang, Li Fei-Fei, Jiayuan Mao, and Jiajun Wu. Embodied agent interface: Benchmarking LLMs for embodied decision making. In *Advances in Neural Information Processing Systems*, volume 37, pages 100428–100534, 2024.

[30] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Paul Saldyt, and Anil B. Murthy. Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 22895–22907, 2024.

[31] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[32] Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. In *Advances in Neural Information Processing Systems*, volume 36, pages 79081–79094, 2023.

[33] Mark Adamik and Paolo Forte. PlanOwl: Automated PDDL files generation from OWL ontologies and visual language models. *Proceedings of the AAAI Symposium Series*, 7(1):634–643, 2025.

[34] Hong Lu, Pierrick Lorang, Timothy R Duggan, Jivko Sinapov, and Matthias Scheutz. Novelty adaptation through hybrid large language model (LLM)-symbolic planning and LLM-guided reinforcement learning. *arXiv preprint arXiv:2603.11351*, 2026.

[35] Tom Silver, Varun Hariprasad, Reece S. Shuttleworth, Nishanth Kumar, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PDDL planning with pretrained large language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

[36] Elliot Gestrin, Marco Kuhlmann, and Jendrik Seipp. NL2Plan: Robust LLM-driven planning from minimal text descriptions. *arXiv preprint arXiv:2405.04215*, 2024. Presented at the ICAPS 2024 HAXP Workshop.

[37] Xiaotian Liu, Armin Toroghi, Jiazhou Liang, David Courtis, Ruiwen Li, Ali Pesaranghader, Jaehong Kim, Tanmana Sadhu, Hyejeong Jeon, and Scott Sanner. Natural language PDDL (NL-PDDL) for open-world goal-oriented commonsense regression planning in embodied AI. In *The Fourteenth International Conference on Learning Representations*, 2026.

[38] Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. AdaPlanner: Adaptive planning from feedback with language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 58202–58245, 2023.

[39] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652, 2023.

[40] 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. 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, 2023.

[41] Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with LLMs enables open-world multi-task agents. In *Advances in Neural Information Processing Systems*, volume 36, pages 34153–34189, 2023.

[42] Kaiqu Liang, Zixu Zhang, and Jaime Fernández Fisac. Introspective planning: Aligning robots' uncertainty with inherent task ambiguity. In *Advances in Neural Information Processing Systems*, volume 37, pages 71998–72031, 2024.

[43] Allen Z. Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, Zhenjia Xu, Dorsa Sadigh, Andy Zeng, and Anirudha Majumdar. Robots that ask for help: Uncertainty alignment for large language model planners. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 661–682, 2023.

[44] Daniel Honerkamp, Martin Büchner, Fabien Despinoy, Tim Welschehold, and Abhinav Valada. Language-grounded dynamic scene graphs for interactive object search with mobile manipulation. *IEEE Robotics and Automation Letters*, 9(10):8298–8305, 2024.

[45] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551, 2023.

[46] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246, 2006.

[47] Aaron Hurst, Adam Lerer, Adam P. Goucher, et al. GPT-4o System Card. *arXiv preprint arXiv:2410.21276*, 2024.

A World Model Update and OWL Inference Examples

This appendix provides examples of the ask/tell update mechanism and OWL inference behavior described in Section 3.1. In the evaluated setting, the asserted graph represents the fully observed scene state. When triples are loaded or updated, GraphDB applies the declared OWL 2 RL semantics through forward chaining and materializes entailments that subsequent queries read together with the asserted facts. Figure 6 shows the world model before and after OWL reasoning in the Marstons Medium scene, and Table 7 lists representative direct updates together with selected inferred facts and the axiom patterns that support those inferences under actions and symbolic observations.



(a) Explicit (2,908 triples). (b) Reasoned (19,394 triples; 16,486 inferred).

Figure 6: **World Model Before and After OWL Reasoning.** The reasoned view includes inferred facts in addition to explicit ones, making the representation substantially denser.

Table 7: **Representative World Model Updates and Inferred Facts.** For each action or symbolic observation event, the table lists directly edited dynamic-layer triples, selected inferred facts after OWL reasoning, and the axiom patterns responsible for those inferences. + and − denote inserted and deleted triples.

Event	Edited Triples	Inferred Facts	Axiom Basis
Action Effects			

PICK-ONE-HAND	`+ holds(right_hand, toothbrush_213)`
	`- isInsideOf(toothbrush_213, storage_cabinet_55)`
		`+ carries(robot, toothbrush_213)`
		`+ isCarriedBy(toothbrush_213, robot)`
		`+ isHeldBy(toothbrush_213, right_hand)`
		`- contains(storage_cabinet_55, toothbrush_213)`
			Inverse + carry/hold propagation

MOVE (carrying)	`+ robotIsInSpace(robot, exercise_room_20)`
	`- robotIsInSpace(robot, door_1)`
		`+ isInSpace(robot, exercise_room_20)`
		`+ artifactIsInSpace(toothbrush_213, exercise_room_20)`
		`+ isInStorey(toothbrush_213, Floor_A)`
		`+ isInBuilding(toothbrush_213, Marstons)`
		`- artifactIsInSpace(toothbrush_213, door_1)`
		`- isInSpace(toothbrush_213, door_1)`
			Spatial propagation + hierarchy

PLACE-IN	`+ isInsideOf(toothbrush_213, bucket_36)`
	`- holds(right_hand, toothbrush_213)`
		`+ contains(bucket_36, toothbrush_213)`
		`- carries(robot, toothbrush_213)`
		`- isCarriedBy(toothbrush_213, robot)`
		`- isHeldBy(toothbrush_213, right_hand)`
			Containment inverse + release cleanup

Symbolic Observation Update			

NEW OBJECT	`+ type(key_new, Artifact)`
	`+ isInsideOf(key_new, flower_vase_196)`
		`+ contains(flower_vase_196, key_new)`
		`+ isInSpace(key_new, bedroom_8)`
		`+ artifactIsInSpace(key_new, bedroom_8)`
		`+ isInStorey(key_new, Floor_C)`
		`+ isInBuilding(key_new, Marstons)`
		`+ hasArtifact(bedroom_8, key_new)`
		`+ spaceHasArtifact(bedroom_8, key_new)`
		`+ hasArtifact(Floor_C, key_new)`
		`+ hasArtifact(Marstons, key_new)`
			Containment inverse + hierarchy closure

B Agent System Prompts

Prompt notation. The prompts below are excerpted verbatim from the released code, with Markdown emphasis removed. Tokens formatted as [dialogue_history] denote state fields injected at runtime, and [...] denotes omitted prompt text. In Figure 3, [tool_history] groups [scene_query_history] and [pddl_plan_history], and [task_conditions] groups the at_end, always, and sometime conditions.

B.1 Flow Orchestrator Prompt

```
You are the Flow Orchestrator, the central orchestrator of the robot task
planning system.

## CONTEXT
[dialogue_history]
[discovered_objects]
[agent_reports]

## YOUR ROLE
You manage the overall workflow by:
1. Determining user intent: Is this a question or a task?
2. Routing to appropriate agents: Decide which agent to invoke next
3. Providing intelligent responses: Answer questions with context and
   suggest follow-up tasks

## ROUTING
You can route to THREE agents:

### 1. Route to USER (Ask for clarification) [...]
### 2. Route to Scene Explorer (Explore environment) [...]
### 3. Route to Task Formalizer (Define constraints & goals) [...]

## DECISION LOGIC
Step 1: Classify user intent [...]
Step 2: Understand with common sense [...]
Step 3: Check if information is sufficient [...]
Step 4: Review system context [...]
Step 5: Make routing decision [...]

## OUTPUT
Always provide:
1. user_intent: "task_planning" or "question_answering"
2. next_agent: "USER" or "scene_explorer" or "task_formalizer"
3. reason: Clear explanation of why you chose this agent
4. question: (ONLY if routing to USER) The question or answer to provide
   - For question_answering: Include answer + optional task suggestion
   - For task_planning: Include clarification question if needed

Analyze the situation and decide the next agent to invoke.
```

B.2 Scene Explorer Prompt

```
You are the Scene Explorer. Think like someone who is physically in this
space trying to help -- use judgment and common sense, not just keyword
matching. Your goal is to give the planner a coherent, complete picture
of the scene.

## CONTEXT
[dialogue_history]
[routing_reason]
[space_overview]
[scene_query_history]

## TOOLS

### semantic_search(query, reason, space_id, floor_id, object_id)
Natural language search with optional spatial filters.
- reason: one sentence explaining why you are making this call
  (required -- always fill this in).
- space_id: narrow to a specific room (e.g., "kitchen_1"). Do NOT use
  for room/space types -- rooms are not inside other rooms.
- floor_id: narrow to a floor (e.g., "Floor_B"). Also used with
  isConnectedTo to resolve spatial adjacency ("room next to X"). [...]

The hint field describes what the tool found and why -- [...]

### find_path(from_id, to_id, reason)
Returns shortest navigation path between two locations. Use when the task
specifies routing constraints (e.g., "nearest kitchen", "avoid corridor").
reason is required.

## STRATEGY
Be thorough. The task implies a full set of objects -- sources,
destinations, containers, portals, and spatial constraints. [...]

Call tools in parallel whenever possible. Start broad, then narrow:
1. Turn 1: search for all task-relevant objects simultaneously
2. Follow the hint if similar categories are needed or initial results
   are empty
3. Use object_id to find items inside containers once the container is
   identified
4. Call finish_exploration once all objects are found [...]

## EXAMPLES
Example 1: Object placement + container contents [...]
Example 2: Spatial adjacency [...]
Example 3: Portal search [...]

## OUTPUT
finish_exploration(exploration_summary="
Task: <one-line restatement>
Found:
  - <role>: ‘obj_id‘ (category) in ‘space_id‘
  - <role>: ‘door_id‘ connects ‘space_a‘ <-> ‘space_b‘
  ...
Not found: <anything not located>
Notes: <ambiguity resolutions only>
") [...]
```

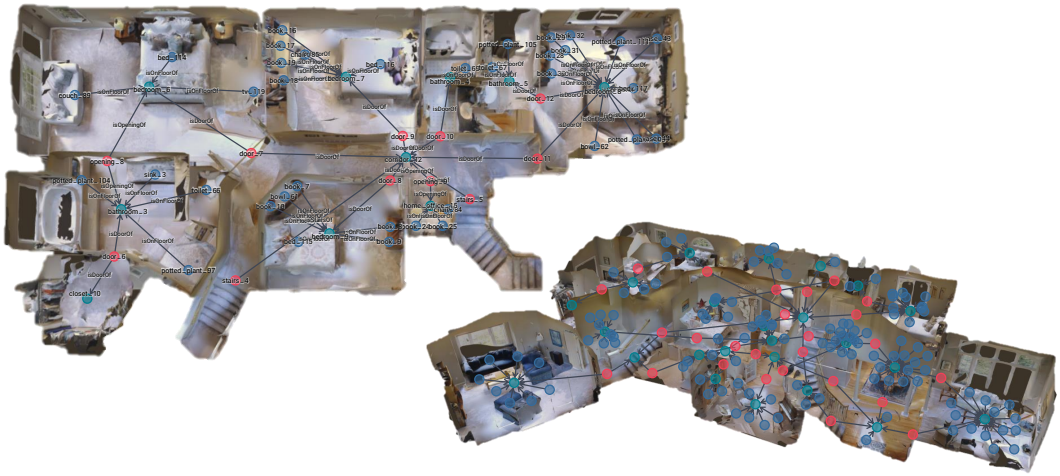
B.3 Task Formalizer Prompt

```
You are the Task Formalizer, responsible for converting user tasks into
PDDL conditions.

## CONTEXT
[dialogue_history]
[routing_reason]
[discovered_objects]

## YOUR ROLE
Convert user task instructions into 3 types of conditions based on PDDL
temporal planning:
1. at_end_condition (REQUIRED): What must be true when the task is complete
2. always_conditions (OPTIONAL): States that must be true throughout the
   entire execution
3. sometime_conditions (OPTIONAL): States that must be achieved at some
   point during execution (including high-level concepts like "make
   coffee", "bake bread")

## AVAILABLE PDDL PREDICATES
- (robotIsInSpace robot <space>) - Robot is in a room/space
- (isOnFloorOf <artifact> <space>) - Artifact on floor of a room/space
- (isInsideOf <artifact> <artifact>) - Artifact inside another artifact [...]
- (isOntopOf <artifact> <artifact>) - Artifact on top of another artifact [...]

## TOOLS
### 1. define_at_end_condition(goal_formula, reasoning) [...]
### 2. define_always_condition(predicate_formula, description) [...]
### 3. define_sometime_condition(predicate_formula, description) [...]

## WORKFLOW
1. Check objects: Do you have all necessary objects? If not ->
     request_more_exploration(reason)
2. Analyze task: What is the user really asking for?
3. Define end goal: What's the final state? -> define_at_end_condition (REQUIRED)
4. Identify constraints: Are there strict constraints that must hold
     during execution? -> define_always_condition
5. Identify intermediate states: Are there intermediate states that must
     occur during execution? -> define_sometime_condition

## CRITICAL RULES
1. Use EXACT object IDs from Available Objects - do NOT invent IDs. [...]
2. define_at_end_condition is MANDATORY -- Your response is INCOMPLETE
   without it. [...]

## OUTPUT
Tool calls in a single response:
  define_at_end_condition(goal_formula, reasoning)            [exactly 1, required]
  define_always_condition(predicate_formula, description)     [0 or more]
  define_sometime_condition(predicate_formula, description)   [0 or more]

Analyze the task and define all necessary conditions now.
```

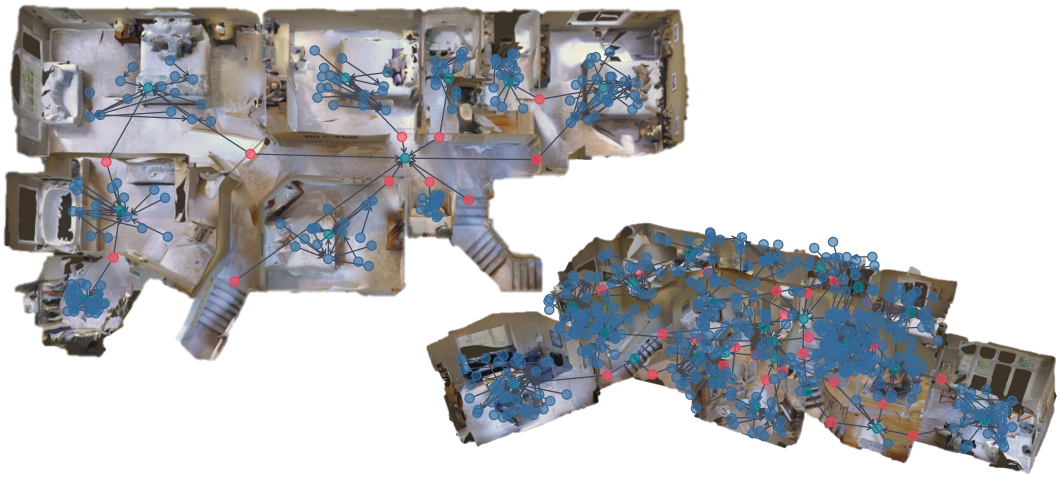
B.4 Planning Manager Prompt

```
You are the Planning Manager - you arrange conditions from Task Formalizer
into ordered subgoals for PDDL planning.

## CONTEXT
[dialogue_history]
[at_end_condition]
[always_conditions]
[sometime_conditions]
[pddl_plan_history]

## YOUR ROLE
CRITICAL: You are an ARRANGER, not a planner. Your job is to:
1. Take the conditions that Task Formalizer already defined
2. Arrange them in the correct execution order as subgoals
3. DO NOT add new intermediate steps or goals beyond what Task Formalizer
   specified [...]

## TOOLS
### 1. pddl_plan(subgoals, task_description) [...]
### 2. approve_plan(reasoning) [...]

## SUBGOAL CREATION
### Step 1: Convert Conditions to Subgoals
- Sometime Conditions -> Each becomes a subgoal [...]
- At-End Condition -> Handle carefully (see Step 2)
- Always Conditions -> Never convert to subgoals, only pass as
  always_conditions parameter
### Step 2: Handle At-End Condition
Default: Keep at_end as ONE subgoal (don't decompose it) [...]
### Step 3: Order the Subgoals [...]
### Step 4: DO NOT Add Extra Steps [...]

## SUBGOAL FORMAT
{
  "goal_state": "(predicate formula)",  # Use ONLY predicates from list above
  "description": "What this achieves",
  "name": "lowercase_name",
  "order": 0  # Execution order
}

## CRITICAL RULES
1. You are an ARRANGER, not a planner - Don't add intermediate steps
   beyond what Task Formalizer defined [...]
7. On retry: NEVER drop conditions - Every sometime condition and the
   at_end condition must always be present as subgoals. Only reorder or
   adjust formulas.

## OUTPUT
ReAct pattern -- Call ONE tool per response:
- First call: pddl_plan(subgoals=[...], task_description="...") [...]
- After pddl_plan result: [...]
  - Plan correctly achieves user intent -> approve_plan(reasoning="...")
  - Any subgoal FAILED, or plan is wrong/inappropriate for the task ->
    pddl_plan(...) again with corrected subgoals [...]

Now arrange the conditions from Task Formalizer into ordered subgoals.
```

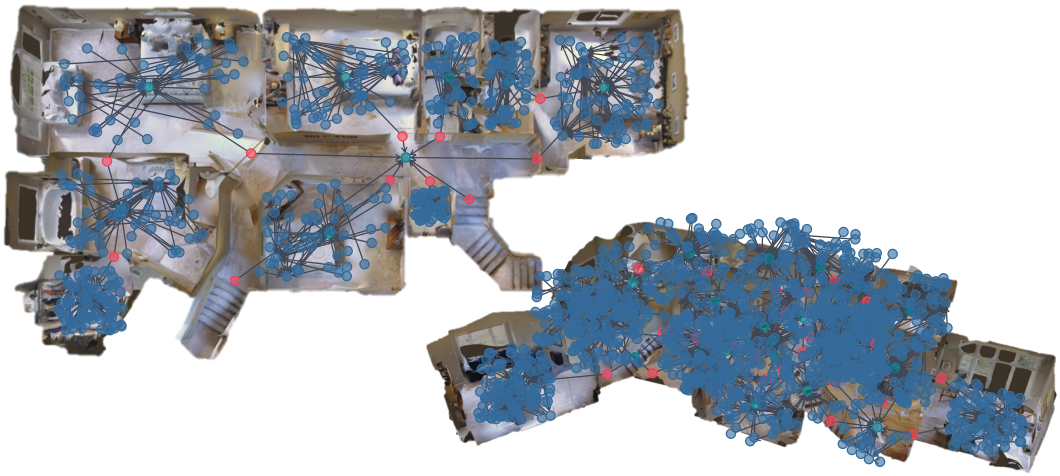
C Dataset Scale Augmentation Visualization



(a) Small. Original source scene graph (121 objects).



(b) Medium. Small with manually added objects (423 objects).



(c) Large. Medium with random augmentation to 3× objects (1269 objects).

Figure 7: **Dataset Scale Augmentation in Muleshoe.** Visualization of the same environment at the three scene scales. Each panel pairs a single-floor view on the left with a full-space view on the right.

D General Instruction Examples

Table 8: **General Instruction Examples for Klickitat.** Thirty instructions from the Small, Medium, and Large splits. S/M/L denote the splits, and gray arrow text shows derived evaluation conditions.

ID	Instruction
S01	I think I left my cell phone in the dining room on Floor B. Can you find it and bring it to the home office on Floor C? → (at_end) cell_phone_61 is in home_office_18.
S02	There should be a bottle in the kitchen on Floor B. Take it out and put it on the couch in the living room on Floor B. → (at_end) bottle_3 is on couch_31 or couch_33.
S03	Move the sports ball from the playroom on Floor C to the home office on Floor C. Never go through the door connecting the lobby and the corridor on Floor B because it is under repair. → (always) the robot does not pass through door_5. (at_end) sports_ball_2 is in home_office_18.
S04	Move the vase in the living room on Floor B to the bathroom on Floor C. Use only your right hand for this task. → (always) left_hand remains empty. (at_end) vase_80 or vase_84 is in bathroom_3, bathroom_4, or bathroom_5.
S05	There is a book in one of the bedrooms on Floor C. Bring it to the living room on Floor B. → (at_end) book_78 is in living_room_22 or living_room_23.
S06	There aren't enough chairs in the dining room connected to the kitchen on Floor B. Can you bring one more chair from the other dining room on the same floor? → (at_end) at least one of chair_24, chair_25, or chair_26 is in dining_room_16.
S07	Move the potted plant from the staircase on Floor B to the lobby. I'm afraid people might bump into it. → (at_end) potted_plant_45 is in lobby_24.
S08	Move the vase from the bathroom on Floor C to the playroom. Use only your right hand. → (always) left_hand remains empty. (at_end) vase_81 is in playroom_25.
S09	I need to move a chair from the dining room on Floor B to the bedroom on Floor A. Please use the stairs leading to Floor A. → (sometime) robot passes through stairs_7. (at_end) at least one chair from chair_18–chair_22 or chair_24–chair_26 is in bedroom_6.
S10	Move one potted plant from the bedroom connected to the closet on Floor C to another bedroom. And make sure to close the door when you leave the room. → (at_end) at least one of potted_plant_46 or potted_plant_47 is in bedroom_6, bedroom_8, or bedroom_9, and door_6 is closed.
M01	Please turn on the coffee machine in the kitchen on Floor B. Also, bring the mug from the tray in the bedroom on Floor C and place it on the coffee machine. → (at_end) coffee_machine_170 is switched on, and mug_71 is on it.
M02	Turn on the TV in the living room on Floor B and gather the cushions scattered around the room onto the sofa. → (at_end) tv_211 is switched on, and cushions cushion_219–cushion_223 are on sofa_208.
M03	I think I left the refrigerator in the kitchen open. Could you take the juice out of it and place it on the dining table? And please close the refrigerator when you are done. → (at_end) juice_357 is on dining_table_112 or dining_table_131, and refrigerator_164 is closed.
M04	Turn on the laptop in the home office on Floor C and take the hole punch out of the storage cabinet onto the desk. → (at_end) laptop_144 is switched on, and hole_punch_323 is on desk_139.
M05	Take the jeans out of the dresser in the closet on Floor C and put them on the armchair in the bedroom. → (at_end) jeans_294 is on armchair_48 or armchair_49.
M06	Take the wine glasses out of the cupboard in the dining room on Floor B and set them on the dining table. → (at_end) wine_glass_121 and wine_glass_122 are on dining_table_112.
M07	Turn on the microwave in the kitchen on Floor B. And if the door leading to the dining room is closed, please open it. → (at_end) microwave_169 is switched on, and door_3 is open.
M08	Cook the chicken from the refrigerator in the kitchen on Floor B in the oven, then serve it on the island table in the kitchen. → (sometime) chicken_353 is inside oven_161 while it is closed and switched on. (at_end) it is on island_table_158.
M09	Heat up the beef from the refrigerator in the kitchen on Floor B in the microwave, then place it on the island table in the kitchen. → (sometime) beef_354 is inside microwave_169 while it is closed and switched on. (at_end) it is on island_table_158.
M10	Wash the bowl from the storage cabinet in the kitchen on Floor A in the dishwasher, then put it back on top of the storage cabinet. → (sometime) bowl_341 is inside dishwasher_156 while it is closed and switched on. (at_end) it is on storage_cabinet_154.
L01	Turn on the microwave in the kitchen on Floor B. But since the corridor is being cleaned, please go through the living room. → (always) the robot does not pass through corridor_12. (at_end) microwave_169 is switched on.
L02	We need one more chair in the dining room on Floor A. Can you bring a chair from the dining room next to the corridor on Floor B? → (at_end) chair_132 or chair_133 is in dining_room_15.
L03	Turn on the TV in the living room on Floor B. But please avoid the stairs connecting the living room and the lobby on Floor B. → (always) the robot does not pass through stairs_4. (at_end) tv_211 is switched on.
L04	There should be a hole punch in the storage cabinet in the home office on Floor C. Take it out and put it on the desk. → (at_end) hole_punch_323 is on desk_139.
L05	The lamps in the bedroom with a bathroom right next to you are on. Could you turn them all off so I can sleep comfortably? → (at_end) lamp_83 and lamp_84 are switched off.
L06	Take the milk from the refrigerator in the kitchen on Floor B and put it on the dining room table. → (at_end) milk_348 is on dining_table_112 or dining_table_131.
L07	Take the jeans out of the closet dresser on Floor C and lay them on the bed. → (at_end) jeans_294 is on bed_53 or bed_73.
L08	Cook the cheese from the refrigerator in the kitchen on Floor B in the oven, then serve it on the island table in the kitchen. → (sometime) cheese_350 is inside oven_161 while it is closed and switched on. (at_end) it is on island_table_158.
L09	Heat up the milk from the refrigerator in the kitchen on Floor B in the microwave, then place it on the island table in the kitchen. → (sometime) milk_348 is inside microwave_169 while it is closed and switched on. (at_end) it is on island_table_158.
L10	Wash the cup from the storage cabinet in the kitchen on Floor A in the dishwasher, then put it back on top of the storage cabinet. → (sometime) cup_343 is inside dishwasher_156 while it is closed and switched on. (at_end) it is on storage_cabinet_154.

E Experimental Protocol Details

E.1 Controlled Implementation Settings

This section summarizes the implementation settings used in the main comparison. Across methods, we use the same task instances, scene facts, initial symbolic states, LLM setting, primitive action set, and symbolic evaluation procedure. Differences are kept only where they are part of the original method design.

LLM setting. All methods use GPT-4o with temperature 0; ONTOPLAN and its ablations use the `gpt-4o-2024-08-06` snapshot. In addition, the `semantic_search` function of ONTOPLAN embeds object categories with OpenAI `text-embedding-3-small` (256 dimensions) and retrieves them with an HNSW index. The world model is served by GraphDB 10.7 with the OWL 2 RL ruleset. All LLM calls are made through the OpenAI API, and GraphDB and Fast Downward run locally on a single CPU workstation.

Scene representation. All methods start from the same scene facts and initial symbolic state. The scene facts include spaces, connectivity, object instances, object locations, object states, and the initial robot state. SayPlan and DELTA receive these facts as serialized text. ONTOPLAN loads the same facts into the ontology-grounded world model. Thus, the comparison varies the scene interface, not the underlying scene information. All scene facts are instantiated before any LLM call through deterministic mappings from the benchmark scene descriptions. None of the evaluated methods receives raw sensor input, and no LLM or learned perception model is used to construct the initial symbolic state.

Primitive action set. All methods use the same 16 primitive actions for task planning and evaluation: move, access, open/close for items and doors, power-on/power-off, and one-/two-handed variants of pick, place-to-location, place-in, and place-on. SayPlan is prompted to generate plans using this primitive action set. DELTA and ONTOPLAN use the same action set through the shared PDDL domain. All returned plans are evaluated with the same action names, preconditions, effects, and costs.

PDDL planning setup. ONTOPLAN uses the shared PDDL domain directly through the *PDDL plan* tool. Following its original design, DELTA receives the same domain as reference context and generates its own planner-facing domain and problem files. Both methods use Fast Downward with `lazy_wastar`, the FF heuristic, and search weight 2.

Exploration and planning calls. SayPlan and ONTOPLAN are each allowed up to 20 scene-exploration LLM calls and 5 plan-generation calls. DELTA follows its fixed workflow and does not use repeated exploration or plan-generation calls.

E.2 Symbolic Evaluation Procedure

All returned action sequences are evaluated with the same symbolic evaluator. Starting from the task initial symbolic state, the evaluator applies actions sequentially, checks each action's preconditions, and updates the symbolic state using the shared action effects. Execution stops after the final action or at the first precondition failure. `always` conditions are checked in the initial state and after every action, `sometime` and `ordering` conditions are tracked over the executed actions, and `at_end` is checked in the reached state. The reached state and execution trajectory are then used to compute the metrics defined in Section 4.2.

F Statistical Analysis

This section reports paired bootstrap 95% confidence intervals for the general-task comparisons with the baselines and for the ablation comparisons. For the baselines, we report ONTOPLAN-minus-baseline differences in the four binary performance metrics from Table 3: *Plan Returned*, *Plan Executable*, *Goal Reached*, and *Task Success*. All intervals are computed using bootstrap resampling stratified by scene scale.

F.1 Analysis Setup

Each bootstrap resample draws 50 tasks with replacement from each scene scale, preserving the benchmark's 1:1:1 Small/Medium/Large composition. We use 10,000 bootstrap resamples. Within each resample, the same task instances are used for ONTOPLAN and each baseline, yielding paired bootstrap 95% confidence intervals. For each metric, we report the rate difference (ONTOPLAN minus baseline). Confidence intervals are computed from the 2.5th and 97.5th percentiles of the bootstrap distribution. This analysis quantifies task-sampling uncertainty while keeping the scene-scale composition fixed.

F.2 Paired Comparisons

Table 9 reports these intervals for ONTOPLAN versus each baseline across the four performance metrics.

Table 9: **Paired Bootstrap 95% Confidence Intervals for General-Task Performance.** Intervals are computed using bootstrap resampling stratified by scene scale. Differences are computed as ONTOPLAN minus baseline across the 150 matched general-task instances.

Baseline	Metric	Difference (ONTOPLAN − baseline)	95% CI
SayPlan	Plan Returned	0.33	[0.25, 0.41]
	Plan Executable	0.61	[0.53, 0.68]
	Goal Reached	0.58	[0.49, 0.66]
	Task Success	0.61	[0.53, 0.69]
DELTA	Plan Returned	0.55	[0.47, 0.63]
	Plan Executable	0.83	[0.77, 0.89]
	Goal Reached	0.73	[0.65, 0.80]
	Task Success	0.75	[0.68, 0.82]

F.3 Ablation Comparisons

Table 10 applies the same paired, scale-stratified bootstrap to the full system and the two ablation variants in Table 6, together with exact McNemar tests on per-task Task Success.

Table 10: **Paired Comparisons Between the Full System and the Ablation Variants.** Differences are computed as full system minus variant across the 150 matched general-task instances. Discordant pairs count tasks solved only by the full system and only by the variant.

Variant	Task Success difference	95% CI	Discordant pairs	McNemar p
w/o *Scene query*	0.02	[−0.04, 0.09]	13 / 10	0.68
w/o *PDDL plan*	0.57	[0.49, 0.65]	87 / 1	$< 10^{-24}$

G Failure Analysis of OntoPlan

OntoPlan fails 17 of the 150 general tasks. Every returned plan executes without a precondition failure (one failed task returns no plan), and the *PDDL plan* tool succeeds on its first call in 145 of the 150 tasks. Comparing the conditions written by the Task Formalizer with the benchmark conditions, Table 11 groups the failures by their origin.

Table 11: **Failure Categories of OntoPlan on the General Tasks.** Tasks are given as environment and task ID.

Category	Count	Tasks
Referring expression grounded to the wrong object or place	13	Klickitat M03, S10; Lindenwood L08, L09, M10, S08; Marstons L03, L04, L05, L07, M10, S10; Muleshoe L07
Part of the goal omitted	2	Lakeville M07; Marstons S07
Trajectory constraint omitted	1	Klickitat L01
No valid goal produced	1	Muleshoe M06

Grounding errors include choosing a different instance of the right category (e.g., `dining_table_58` instead of `dining_table_57` in Lindenwood S08) and resolving a description to the wrong space (e.g., "the study" to `bedroom_7` instead of `home_office_21` in Marstons L04). Omissions include leaving out some of the doors to open in "open all closed doors" instructions and not formalizing the instruction to avoid the corridor in Klickitat L01. These errors arise in instruction interpretation and scene grounding rather than in planning, which motivates verification of grounded referents and formalized conditions as future work.

H Unreturned Plans Across Methods

Table 12 attributes every general task for which a method returned no plan to the stage at which it stopped. OntoPlan and DELTA obtain plans from Fast Downward with the same configuration and a 60 s limit per subgoal, whereas SayPlan generates plans with the LLM after scene-graph search, so their failures arise at different stages. For DELTA, we take the first sub-problem it could not solve and check whether it parses, whether its predicates match the domain in name, arity, and argument types, and whether every object it mentions is declared. If the sub-problem is well formed but unsolvable, we add the true scene facts over its declared objects, and then the missing scene objects with their facts, and record which addition makes it solvable. For SayPlan, we identify the condition that ended its scene-graph search. No unreturned plan was caused by the planner time limit.

Table 12: **Causes of Unreturned Plans on the 150 General Tasks.** Counts are numbers of tasks.

Method	Cause	Count
OntoPlan	No valid goal produced	1
DELTA	Objects used in the sub-problem not declared	61
	Initial-state facts needed to reach the subgoal missing	18
	Predicate applied to arguments of the wrong type	4
	Free variable in the goal	1
SayPlan	Invalid scene-graph search command	44
	Search budget of 20 calls exhausted	4
	Context window of the LLM exceeded	3

DELTA's failures stem from incomplete or malformed planner-facing specifications. In 48 of the 61 tasks with undeclared objects, the undeclared objects are rooms, doors, or stairs that the generated problem refers to without declaring; the four wrong-type cases place an object inside a room or inside a surface that cannot contain objects. SayPlan's failures occur during scene-graph search: in 30 of the 44 tasks with an invalid command, the LLM tries to expand or contract an object node instead of a room, and in the remaining tasks it issues an action or a malformed command. The three context-window failures occur in Large scenes, where the accumulated search prompt exceeds the 128k-token context window of GPT-4o.

I Full Results on Ambiguous Instructions

Table 13: **Per-Instruction Results on Ambiguous Instructions.** Outcomes for all 10 ambiguous instructions with the fixed automatic reply versus direct user replies to clarification questions. ✓/✗ denote whether the goal is reached, and gray arrow text summarizes the interpreted user intent.

ID	Instruction	Goal Reached		Interaction Log
		Auto	User	
A01	Now that the guests have left, put the coffee cups in the living room back where they originally were. → return coffee_cup_228, coffee_cup_229, and coffee_cup_230 to kitchen_20.	✗	✓	**OntoPlan:** Where were the coffee cups originally placed in the living room? **User:** Kitchen on Floor_B.
A02	Bring me the new juice we bought yesterday so I can try it in the Floor A living room. → move juice_357 to living_room_21.	✓	✓	**OntoPlan:** Which juice is the new one you bought yesterday? Is it juice_357 (orange plastic) in kitchen_20 or juice_381 (green wooden) in bathroom_3? **User:** I mean juice_357.
A03	Bring me the vase Anna gifted me to my bedroom so I can take a photo. → move vase_373 to bedroom_8.	✗	✓	**OntoPlan:** Is vase_373 (located in living_room_21) the vase Anna gifted you? Also, which bedroom is yours: bedroom_6 (Floor_A), bedroom_7, bedroom_8, or bedroom_9 (Floor_C)? **User:** Yes, it is. My bedroom is number 8.
A04	Bring the book I was reading yesterday from the living room to my room. → move book_203 to bedroom_8.	✗	✓	**OntoPlan:** Which book were you reading yesterday? book_202 (white, living_room_22), book_203 (beige, living_room_22), or book_204 (beige, living_room_22)? And which bedroom is your room? bedroom_6 (Floor_A), bedroom_7, bedroom_8, or bedroom_9 (Floor_C)? **User:** The book is book_203, and my room is bedroom_8.
A05	We finished eating, so please clean up the desk in the dining room. → move sugar_bowl_130, pepper_shaker_129, and salt_shaker_128 into cupboard_119.	✗	✓	**OntoPlan:** I found sugar_bowl_130, pepper_shaker_129, and salt_shaker_128 on desk_127 in the dining room. What should I do with each item after cleaning them up? **User:** Move all of them to cupboard in same room.
A06	Bring the thing that broke while cooking yesterday to the lobby. → move oven_161 to lobby_24.	✗	✓	**OntoPlan:** Which object broke while cooking yesterday? Is it the kitchen_drawer_167 in kitchen_20 or the vegetable_basket_355 in kitchen_20? **User:** It's oven.
A07	Put back the shoes used as a design sample in the home office to their original place. → return shoes_149 to bookshelf_140.	✗	✓	**OntoPlan:** Which closet should I put the shoes back in? closet_10 or closet_11 on Floor_C? **User:** Return it to bookshelf.
A08	There are two pillows in John's room. He doesn't like one of them. Swap it with one of the pillows in my room. → swap pillow_82 and pillow_66 between bed_53 and bed_73.	✗	✗	**OntoPlan:** There are no pillows found in John's room (playroom_25). Which pillow from your room (bedroom_7) would you like to swap? **User:** John's room is not playroom. His room is bedroom. Please move the pillow.
A09	Plates on the floor are dangerous if they break, so put them back where they originally were. → move plate_137 and plate_110 into cupboard_134.	✗	✗	**OntoPlan:** Where were the plates originally placed? **User:** Put both into cupboard in same room.
A10	Please tidy up the desk in the living room. → move mug_205, mug_206, book_203, and book_204 into wall_cabinet_191.	✗	✓	**OntoPlan:** I found book_203, book_204, mug_205, and mug_206 on desk_193 in the living room. Where should I place each item after tidying? **User:** Move mug_205, mug_206, book_203, and book_204 into any wall cabinet.

J Full Results on Infeasible Instructions

Table 14: **Per-Instruction Results on Infeasible Instructions.** Outcomes for all 10 infeasible instructions in the Klickitat Medium environment. Gray arrow text summarizes the infeasibility rationale.

ID	Instruction	Response Type	System Output
I01	I want to skateboard again after a long time. Could you find the skateboard in the living room on the B floor and bring it to the lobby? → no skateboard in the environment.	Clarification Request	*We couldn't find the skateboard in either of the living rooms on the B floor. Could you please confirm if it might be in another location or provide more details about where it might be?*
I02	I need an extra chair for the guests. Please pick up the chair in the kitchen on the A floor and bring it to the lobby. → no chair in the kitchen on Floor A.	Clarification Request	*No chairs were found in the kitchen on Floor A. Could you please specify another location where I might find a chair?*
I03	I'm thirsty after doing laundry. Could you move to the laundry room and find the juice, then bring it to the kitchen on the B floor? → no laundry room in the environment.	Clarification Request	*The juice is already located in the kitchen on Floor B. Do you need anything else, or is there another task you intended?*
I04	I want to watch TV in the bedroom. Please bring the remote control that is on the side table in the living room on the B floor to the bedroom on the C floor. → no remote control on the side table (none exists in the environment).	Clarification Request	*We couldn't find the remote control on the side tables in the living rooms on the B floor. Could you please confirm its location or provide more details?*
I05	I have a business trip coming up. Could you take the briefcase out of the storage cabinet in the home office on the C floor and bring it to the lobby on the B floor? → no briefcase in the storage cabinet (none exists in the environment).	Clarification Request	*Could you provide more details about the briefcase? For example, its color, size, or any distinguishing features that might help us locate it.*
I06	I'd like to read before bed. Please turn on the book in the bedroom on the C floor. → *book* is not an operable device.	Misdirected Plan	**Plan interpretation:** Reinterpreted as a lamp-related request and generated a plan to turn on `lamp_85`.
I07	The kitchen is too cramped. Could you put the refrigerator inside the microwave in the kitchen on the B floor? → physical containment impossible due to size constraints.	Clarification Request	*It seems impractical to put the refrigerator inside the microwave due to size constraints. Could you clarify what you would like to achieve in the kitchen? Perhaps we can find another way to manage the space.*
I08	I need that book for the meeting tomorrow. Could you bring the book from the living room on the B floor to the bedroom on the C floor? The baby is on the stairs, so please don't use the stairs. → only feasible path requires stairs, which is disallowed.	Clarification Request	*The only available path to move the book involves using the stairs, which you asked to avoid. Would you like to proceed with this path, or do you have any other instructions?*
I09	I have a meeting on the 4th floor. Could you go to the 4th floor and find the laptop, then bring it to the lobby on the B floor? → the building has no 4th floor (Floors A–C only).	Clarification Request	*I couldn't find any laptops on the 4th floor. Could you please confirm if it's in a different location or provide more details?*
I10	I need five books for the book club meeting. Please bring all five books from the living room on the B floor to the kitchen. → only three books available in the specified source.	Clarification Request	*Only three books were found in the living room: `book_202`, `book_203`, and `book_204`. We couldn't locate the remaining two books. Do you have any additional instructions or should we proceed with just these three books?*

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Introduction state the main claims—an ontology-grounded world model aligned with the PDDL domain, a tool-using workflow for grounding and symbolic planning, and improved task success and token efficiency—and the method and experiments support them. The paper also makes the scope explicit: fully observable symbolic indoor environments under controlled benchmark evaluation (Abstract; Sections 1, 3, 4, and 6).

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 6 discusses the main limitations actually emphasized in the paper: fully observable symbolic environments and conclusions limited to controlled planning quality rather than end-to-end real-world autonomy. Section 4 further makes the controlled benchmark scope explicit and notes that the benchmark tasks are manually authored.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper presents no theorems, lemmas, or formal proofs; its contributions are methodological and empirical (Sections 3–5).

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper specifies the evaluation protocol, metrics, shared LLM and planner settings, scene-scale construction, and agent prompts needed to reproduce the main results (Sections 3–5; Appendix B; Appendix C; Appendix E).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code, benchmark tasks, environments, and evaluation scripts are released at `https://github.com/namhyeongwoo/OntoPlan` (linked from the project page in the abstract), with installation and reproduction instructions; baseline implementations are not included. The paper and appendix additionally document the dataset construction, evaluation protocol, implementation settings, and prompt structure needed for faithful reproduction (Abstract; Section 4; Appendix B; Appendix E).

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4 and Appendix E specify the environments, scene scales, task subsets, scoring checks, shared LLM setting, planner backend, and search configuration. Training-specific details such as optimizer choice are not applicable because the method is not trained; agent prompts are documented in Appendix B.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The paper reports paired bootstrap 95% confidence intervals for the main general-task comparisons and the ablations, together with exact McNemar tests on per-task Task Success for the ablations. Appendix F describes the stratified paired bootstrap procedure, 10,000 resamples, and percentile-based intervals for the four main performance metrics.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All LLM calls are made through the OpenAI API, and the paper reports the token usage of every method (Sections 4–5). The world model (GraphDB) and the planner (Fast Downward) run locally on a single CPU workstation (Appendix E).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

Justification: The study uses symbolic indoor benchmarks, involves no human subjects or crowdsourcing, and does not release high-risk generative models or scraped datasets (Sections 4 and 6). To the best of our knowledge, it conforms to the NeurIPS Code of Ethics.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [N/A]

Justification: The paper studies planning quality in fully observable symbolic indoor benchmark environments and confines evaluation to controlled symbolic experiments rather than real-world deployment or real-robot execution studies (Sections 4–6). We therefore treat direct societal impacts of the submitted work as not applicable.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release high-risk generative models, scraped datasets, or similar assets. It studies a robot task-planning framework in symbolic indoor environments rather than releasing assets with elevated misuse risk (Sections 3–5).

Guidelines:

- The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The 3D Scene Graph dataset [16] is used under its non-commercial research-only license, and the derived scene files we release are distributed under the same terms. Fast Downward [46] is used under GPLv3, and GPT-4o and the OpenAI embedding model are accessed through the OpenAI API under the OpenAI Services Agreement.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The released repository documents the benchmark tasks and their evaluation conditions, the environment files at the three scales, their construction, and the license terms of the source data, alongside the paper's description (Section 4; Appendix D; Appendix C).

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The study uses authored benchmark tasks in symbolic indoor environments and includes no crowdsourcing or human-subject experiments (Section 4).

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: No human participants or crowdsourced data were involved, so IRB approval or an equivalent review was not applicable (Section 4).

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLM usage is central to the method and is documented in the introduction, method, experimental setup, and appendix: GPT-4o-based agents interpret instructions, query tools, formalize constraints, and coordinate symbolic planning (Sections 1, 3, and 4; Appendix B).

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.